



Pyxis Project Manager User's Manual

for the Pyxis Custom Design Platform

Software Version 10.5_1

© 1990-2015 Mentor Graphics Corporation
All rights reserved.

This document contains information that is proprietary to Mentor Graphics Corporation. The original recipient of this document may duplicate this document in whole or in part for internal business purposes only, provided that this entire notice appears in all copies. In duplicating any part of this document, the recipient agrees to make every reasonable effort to prevent the unauthorized use and distribution of the proprietary information.

This document is for information and instruction purposes. Mentor Graphics reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Mentor Graphics to determine whether any changes have been made.

The terms and conditions governing the sale and licensing of Mentor Graphics products are set forth in written agreements between Mentor Graphics and its customers. No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Mentor Graphics whatsoever.

MENTOR GRAPHICS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

MENTOR GRAPHICS SHALL NOT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF MENTOR GRAPHICS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

U.S. GOVERNMENT LICENSE RIGHTS: The software and documentation were developed entirely at private expense and are commercial computer software and commercial computer software documentation within the meaning of the applicable acquisition regulations. Accordingly, pursuant to FAR 48 CFR 12.212 and DFARS 48 CFR 227.7202, use, duplication and disclosure by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in the license agreement provided with the software, except for provisions which are contrary to applicable mandatory federal laws.

TRADEMARKS: The trademarks, logos and service marks ("Marks") used herein are the property of Mentor Graphics Corporation or other parties. No one is permitted to use these Marks without the prior written consent of Mentor Graphics or the owner of the Mark, as applicable. The use herein of a third-party Mark is not an attempt to indicate Mentor Graphics as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A current list of Mentor Graphics' trademarks may be viewed at: www.mentor.com/trademarks.

The registered trademark Linux® is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777
Telephone: 503.685.7000
Toll-Free Telephone: 800.592.2210
Website: www.mentor.com
SupportNet: supportnet.mentor.com/

Send Feedback on Documentation: supportnet.mentor.com/doc_feedback_form

Table of Contents

Chapter 1	
Overview	17
The Pyxis Project Manager Tool	17
Pyxis Project Manager Windows	19
Session Window	20
Navigator Window	20
Tools and Toolbox Window	21
Project Navigator Window	22
Configuration Window	23
Monitor Window	23
Trash Window	24
Types Window	25
Information Window	26
The Change/ Fix References Window	26
Chapter 2	
Key Concepts	31
A Design Object	31
Filesets	34
Types	35
Versions	37
References	39
Properties	41
Common Operations on Design Objects	42
Special Design Objects	51
Pyxis Project Manager Display	54
Navigation	55
Navigator	55
Navigator Components	59
Reference Navigation	62
The Object Browser Dialog Box	63
Pathnames in the Pyxis Project Manager	69
Design Management with Location Maps	71
The Location Map	73
Working with Location Maps	77
Following References	83
The MGC Working Directory	85
Moving the MGC_HOME software tree	87
Working Without a Location Map	87
Location Map Administration	89
Tool Invocation	102
The Tools Window	102

Startup Files	112
The Default Startup File	114
The Toolbox Search Path Startup File	115
Startup File Search Order	116
Configuration Management Concepts	117
Creating a Configuration	118
Managing Configurations in Batch Mode	122
Configuration Monitoring	122
Configuration Operations	123
Integrated Design Management (iDM)	131
Design Management with iDM	131
 Chapter 3	
Operating Procedures	141
Open and Close the Pyxis Project Manager	141
Pyxis Project Manager's Graphical Interface	142
Mouse Functions	142
Session Window	146
Updating a Window	152
Setup Pulldown Menu Options	153
Design Navigation	157
Location Map	157
Open a Navigator	157
Search For a Design Object	159
Tool Invocation	162
Invoking from the Tools Window	162
Invoking with the MGC_INVOKE_SETUP Variable	163
Invoking from a Navigator	165
Getting Information About a Tool	165
Changing the Toolbox Search Path	166
Design Objects	169
Opening a Design Object	169
Getting Information about a Design Object	170
Copy, Move, and Rename a Design Object	171
Delete a Design Object	187
Change the References of a Design Object	190
Control Access to a Design Object	190
Creating a Basic Container Design Object	193
File System Objects	194
Versions	197
Showing the Versions of a Design Object	197
Getting Information about a Version	198
Opening a Version	199
Deleting from the Version Window	200
Deleting Multiple Back Versions	201
Copying a Version	202
Freezing and Unfreezing a Version	203
Showing Version Properties	203

Table of Contents

Changing Version Depth	204
Reverting to a Previous Version	205
References	205
Showing the References of a Design Object	206
Getting Information about a Reference	207
Add and Delete a Reference	208
Change References	210
Convert Hard References to Soft References	214
Checking for Broken References	217
Fixing Reference Problems Due to Viewpoints and Part Interfaces	220
Working with Reference Properties	220
Design Object Properties	222
Showing the Properties of a Design Object	222
Adding an Object Property	222
Deleting a Property	223
Changing an Object Property	223
Changing Reference Properties	224
Change Your Session Setup	224
Save Session Setup Settings	224
Specifying the Default File Editor	225
Changing the Appearance of Iconic Windows	226
Specifying Startup Windows	228
Specifying Session Defaults	231
Specifying Navigator Filters	232
Specify Configuration Monitoring	236
Managing Designs	238
Creating a Design Configuration	238
Viewing a Design Configuration	252
Opening an Existing Design Configuration	255
Get Information About a Design Configuration	256
Maintain the Configuration Hierarchy	259
Retargeting Configuration Entries	260
Copy a Design Configuration	262
Releasing a Design Configuration	266
Changing the References of a Design Configuration	272
Freezing and Unfreezing a Design Configuration	273
Lock a Design Configuration	274
Deleting a Design Configuration	275
Archive and Restore Designs	276
Managing Designs Using iDM	279
Copying Design Objects	280
Moving Design Objects	283
Deleting Design Objects	287
Changing References on Design Objects	288
Display a Component Hierarchy	290
Displaying Component Information	293
Design Management in the Operating System Shell	302
Using the Nodisplay Mode	302
Pyxis Project Manager Shell Commands	304

Customizing Your Design Management Environment	306
Installing Process Design Kits (PDKs)	307
ICanalyst Projects	308
Creating a New ICAnalyst Project	308
Chapter 4	
Project Navigator	311
Project Navigator Invocation	311
Project Structure	312
Project	313
External Library	314
Technology Library	315
Library	315
Category	315
Technology Category	316
Cell	316
Project Navigator Graphical Interface	316
Project Navigator Toolbar	316
Project Navigator Window Display	318
Project Navigator Popup Menu	320
Project Navigator Hierarchy	320
Project Hierarchy	321
Opening a Hierarchy	321
Copying a Design Hierarchy	323
Viewing the recent history of the hierarchy	326
Closing a Hierarchy	326
Project Data and Components	327
Creating a Technology Library	327
Creating a Technology Configuration	328
Creating a Technology Category	331
Creating a New Project	331
Creating a New External Library	332
Creating a New Library	334
Creating a New Category	335
Creating a New Cell	335
Create View Objects	336
Creating a New Layout	336
Creating a New Symbol	337
Creating a New Schematic	338
Specifying a New Language Source	338
Manage Data for a Project	339
Copying Design Objects	340
Moving Design Objects	343
Renaming Design Objects	346
Deleting Design Objects	347
External Libraries and Logic Libraries	348
Including External and Logic Libraries	349
Removing External and Logic Libraries	350

Table of Contents

Finding External Dependencies	350
Manage Technology Settings.....	352
Changing the Technology Settings.....	352
Validating the Technology Settings	353
Viewing the Technology Settings.....	354
Chapter 5	
Revision Control	357
Revision Control Terminology	357
User Credentials.....	359
Revision Control Environment Variables and Setup.....	359
Environment Variables	360
Setting Up the ClioSoft Environment.....	360
Setting Up the Subversion Environment.....	362
Pyxis Project Setup.....	362
Object Status States	363
Object Status Flow	365
Viewing Object Status	366
Filtering Objects	369
Revision Control Menu in Pyxis Project Manager	371
ClioSoft Revision Control Procedures.....	374
Adding a Project to a Repository (SOS).....	374
Creating a Work Area (SOS)	376
Changing the Revision Search Order (RSO)	377
Using the Revision Log (SOS)	379
Applying Snapshots	381
Tagging Objects	382
Setting a Branch	383
Subversion Revision Control Procedures	385
Adding a Project to a Repository (SVN)	385
Creating a Work Area (SVN)	386
Using the Revision Log (SVN).....	387
Common Revision Control Procedures	390
Using Automatic Check In	390
Checking Out Objects.....	391
Canceling an Object Check Out	391
Checking In Objects	393
Updating an Object	395
Managing Hierarchy	395
Data Management.....	397
Subversion Administrator Information	398
Subversion Client	398
Creating a Repository	398
Apache Versus SVNServer	399
Integrating Custom Type Data with Revision Control	403
Qualification Scripts and Revision Control Integration.....	403
ClioSoft Troubleshooting	405

Chapter 6	
OpenAccess Import/Export	407
OpenAccess Import Prerequisites	407
Using OpenAccess Schematic Import	407
Using OpenAccess Layout Import	410
OpenAccess Export Prerequisites	411
Using OpenAccess Schematic Export	411
Using OpenAccess Layout Export	413
OpenAccess Via Export/Import	414
OA Via Export	414
OA Via Import	414
Chapter 7	
Function Summary	417
Summary of All Functions	417
Interactive Function Summary	433
Design Object Toolkit Function Summary	440
Configuration Management Toolkit Function Summary	445
Tool Viewpoint Function Summary	448
iDM Interactive Function Summary	449
iDM Toolkit Function Summary	451
Chapter 8	
Language Flow	453
Language Integration for Compilation	453
Prerequisites for Working with Language Files	454
Invoking the Language Editor from Pyxis Project Manager	454
Setup Preferences Dialog Box (Language Interface Panel)	457
Specifying Compilation Options for a Project	458
Specifying Alternate Compilation Settings for an HDL File	460
Compiling HDL source file(s)	462
Model Registration or Symbol Generation	463
Specifying the Default Registration Settings	464
Project Registration Options Dialog Box	465
Default Symbol Layout	467
Specifying the Symbol Layout Options	467
Symbol Layout Properties Dialog Box	469
Viewing a generated symbol	471
Registering a Model from a Source File	472
Registering All Models from a Source File	474
Registering a Model from a Compiled Library	475
Mapping or Unmapping Symbol Pins to the Model	477
Map Pins Dialog Box	477
Mapping or Unmapping Symbol Properties to the Model	480
Map Properties Dialog Box	480
Checking the Language Views	483
INI Mappings	484
Managing INI Mappings	484

Table of Contents

Language Import	487
Importing an HDL file	487
Importing a SPICE file	490
Appendix A	
Troubleshooting	493
Salvage a Damaged Design Object	493
Finding a Design Object's Name and Type	494
Recognizing When a Design Object Is Damaged.....	495
The Salvage Process	496
Salvaging by Deleting Edit Files	498
Salvaging with Edit Recovery	498
Protecting Your Data During a Salvage	499
Problems with Type Registries	503
Determining That a Type is Missing	503
Adding a Type to Your Environment.....	504
Adding a Type through the Type Window	505
Adding a Type with the MGC_TYPE_REGISTRY Shell Variable.....	506
Appendix B	
Pyxis Project Manager Standard Properties	507
Design Object Properties	507
Appendix C	
Design Object Filesets.....	509
Data Files	509
Required and Optional Files	510
Files Without Suffixes	510
Attribute Files	510
Version Files	511
Container Filesets	512
Metadata Files	513
Appendix D	
Pyxis Project Manager Design Object Types	517
Design Objects	517
Appendix E	
Design Object Types and Tool Icons.....	519
Design Object Types	519
Pyxis Custom Design Flow Tool Icons	520
Appendix F	
Releasing Latched Data in Pyxis Project Manager	523
Constraints for Releasing Latched Viewpoints	523
Building and Releasing a Configuration in Latched Mode	524
Latched Mode in the Configuration Window	525

Latched Mode in the Navigator Window	525
Appendix G	
User Interface and Scopes	527
Windows and Scopes	527
Appendix H	
Managing Location Maps	529
Design Management with Location Maps	529
Master Location Maps	529
Project Location Maps	530
Individual Location Maps	531
Location Map Description	532
Location Map Version	532
Comment Lines	533
Soft Prefix Definitions	534
Included Location Maps	537
How the DDMS Uses the Location Map	538
Location Map Search Hierarchy	538
The Location Map and Existing References	540
The Location Map and New References	541
Location Map Creation	542
Template File Location	542
Guidelines for Creating Soft Prefixes	543
Determining Existing Hard Pathnames and Soft Prefixes	544
Controlling Environment Variables	545
Location Map Network Administration Examples	547
Appendix I	
Importing Design Data	551
Importing a Technology Design Kit	552
Importing an IC Studio Project	553
Importing an IC Studio Library	557
Importing Classic Data	559
Importing External Library	562
Importing Custom View	563
Appendix J	
EDIF Converter Quick Start Guide	565
EDIF Converter	565
Acronyms	565
Usage	565
Main Functionality	567
Features Provided	567
About Schematic Language Generation in the NCF	568
Assumptions	568
Command File	569
Command Syntax	570

Table of Contents

Sample Command Files	576
NCF Generation	579

Glossary

Third-Party Information

End-User License Agreement

List of Figures

Figure 1-1. The Pyxis Project Manager Graphical Interface	18
Figure 1-2. Navigator Window	20
Figure 1-3. Tools Window	21
Figure 1-4. Toolboxes Window	21
Figure 1-5. Project Navigator Window	22
Figure 1-6. Configuration Window	23
Figure 1-7. Session Monitor Window	24
Figure 1-8. Configuration Monitor Window	24
Figure 1-9. Trash Window	25
Figure 1-10. Types Window	25
Figure 1-11. Information Window	26
Figure 1-12. Change/Fix References Window	28
Figure 2-1. Basic Elements of a Design Object	32
Figure 2-2. Version Pruning and Freezing	38
Figure 2-3. Reference Updating During a Copy	45
Figure 2-4. Reference Updating and Breaking After a Move Command	49
Figure 2-5. Renaming a Component Containing a Symbol	50
Figure 2-6. Large and Small Configuration Object Icons	54
Figure 2-7. Pyxis Project Manager Session Window	55
Figure 2-8. Iconic and List Navigators	57
Figure 2-9. Navigator Components	59
Figure 2-11. Exploring the Design	62
Figure 2-12. Browse for Object Dialog Box	64
Figure 2-13. Sample of a Location Map File	75
Figure 2-14. The Tools Window	103
Figure 2-15. Example of a Toolbox Window	110
Figure 2-16. Toolbox Search Path	111
Figure 2-17. Configuration Window	118
Figure 2-18. Large and Small Configuration Object Icons	118
Figure 2-19. Example of a Release of Multiple Revisions	128
Figure 2-20. Checkpoint of an Evolving Design	129
Figure 2-21. Reference Updating During a Copy	134
Figure 2-22. Reference Updating and Breaking After a Move Command	136
Figure 3-1. Windows Toolbar (Icon Only mode)	148
Figure 3-2. Windows Toolbar (Icon/Text mode)	148
Figure 3-3. Interface Toolbar (Icon Only mode)	148
Figure 3-4. Pyxis Project Manager Toolbar (Icon Only mode)	148
Figure 3-5. Configuration Window Toolbar (Icon Only mode)	149
Figure 3-6. Project Navigator Toolbar (Icon Only mode)	149
Figure 3-7. Pyxis Project Manager Session Window	150

List of Figures

Figure 3-8. Admin Login Dialog Box	154
Figure 3-9. New Type Prompt Bar	154
Figure 3-10. Edit Type Dialog Box	155
Figure 3-11. Iconic and List Navigators	159
Figure 3-12. Example of a Tools Window.	163
Figure 3-13. Example of Toolbox Window	166
Figure 3-14. Copy Object Options dialog box	174
Figure 3-15. Release Object Options Dialog Box	178
Figure 3-16. Release Design Dialog Box	180
Figure 3-17. Move Object Options Dialog Box	184
Figure 3-18. Change Object Name Options Dialog Box	186
Figure 3-19. Delete Confirmation Dialog Box	188
Figure 3-20. Trash Can Icon	188
Figure 3-21. Change Protection Dialog Box	192
Figure 3-22. Version Window	198
Figure 3-23. Example of a Reference Window	207
Figure 3-24. Change/Fix References Dialog Box	211
Figure 3-25. Convert Object References Dialog Box	215
Figure 3-26. Convert Configuration References Dialog Box	216
Figure 3-27. Fix Broken References Dialog Box	218
Figure 3-28. Iconic Navigator Setup Dialog Box	227
Figure 3-29. Setup Preferences Dialog Box Session Tab	229
Figure 3-30. Setup Session Preferences Dialog Box General Tab	231
Figure 3-31. Setup Filters Tab	234
Figure 3-32. Filter Objects for the Active Navigator Dialog Box	236
Figure 3-33. Monitor Status Setup Dialog Box	237
Figure 3-34. Set Build Rules Dialog Box	243
Figure 3-35. Maintain Hierarchy Dialog Box	245
Figure 3-36. Configuration Window, Before and After Building	248
Figure 3-37. Leaf Resolution, Before and After Building	249
Figure 3-38. Referencing Objects Inside of the Container	250
Figure 3-39. Saved Configuration, with Configuration Object Added	252
Figure 3-40. Viewing the Containment Hierarchy	253
Figure 3-41. Viewing the Primary Hierarchy	254
Figure 3-42. Viewing the Secondary Hierarchy	255
Figure 3-43. Large and Small Configuration Object Icons	255
Figure 3-44. MGC Design Management Menu	280
Figure 3-45. Copy Design Object Dialog Box	281
Figure 3-46. Copy Design Object Options Dialog Box	282
Figure 3-47. Move Design Object Dialog Box	284
Figure 3-48. Move Design Object Options Dialog Box	285
Figure 3-49. Delete Design Object Dialog Box	287
Figure 3-50. Delete Design Object Options Dialog Box	288
Figure 3-51. Change Design Object References Dialog Box	289
Figure 3-52. Component Hierarchy Window	291

Figure 3-53. Hierarchy Popup Menu	292
Figure 3-54. Component Window Display	294
Figure 3-55. Component Window with No Model Information Showing	297
Figure 3-56. Component Window Expanded Information	297
Figure 3-57. Process Design Kit Import	307
Figure 3-58. PDK Installation Options	308
Figure 3-59. New ICanalyst Project Dialog Box	309
Figure 3-60. Create ICanalyst Project Options Dialog Box	310
Figure 4-1. Windows Toolbar	312
Figure 4-2. Structure of the Project Hierarchy	313
Figure 4-3. Project Navigator Toolbar (with project as the active design object)	314
Figure 4-4. Project Navigator Window	319
Figure 4-5. Open Hierarchy Dialog Box	322
Figure 4-6. Copy Design Hierarchy Dialog Box	324
Figure 4-7. Copy Design Hierarchy: Options	325
Figure 4-8. Copy Design Hierarchy: Follow References	325
Figure 4-9. Copy Design Hierarchy: Filters	325
Figure 4-10. Create New Technology Library Dialog Box	327
Figure 4-11. New Technology Configuration Dialog Box	329
Figure 4-12. Create New Technology Category Dialog Box	331
Figure 4-13. New Project Dialog Box	332
Figure 4-14. New External Library Dialog Box	333
Figure 4-15. Create New Library Dialog Box	334
Figure 4-16. Create New Symbol Dialog Box	337
Figure 4-17. New Schematic Dialog Box	338
Figure 4-18. New Language Source Dialog Box	339
Figure 4-19. Copy Object Dialog Box	341
Figure 4-20. Preview of Conflicts Dialog Box (for Copy Operation)	342
Figure 4-21. Move Object Dialog Box	344
Figure 4-22. Preview of Conflicts Dialog Box (for Move Operation)	345
Figure 4-23. Rename Object Dialog Box	346
Figure 4-24. Confirmation of the Delete Object operation	348
Figure 4-25. Manage External/Logic Libraries	349
Figure 4-26. Find External Dependencies Dialog Box	351
Figure 4-27. Reference Filters Dialog Box	352
Figure 4-28. Set Technology Dialog Box	353
Figure 4-29. View Technology Dialog Box	355
Figure 5-1. Object Status - Order of Precedence	364
Figure 5-2. Object Status Flow	365
Figure 5-3. Object Status (Tree View) Dialog Box	367
Figure 5-4. Setup Preferences Revision Control Dialog Box	369
Figure 5-5. Revision Control Menu	371
Figure 5-6. Revision Control Pop Up Menu	373
Figure 5-7. Add Project Dialog Box (SOS)	375
Figure 5-8. Create Work Area Dialog Box (SOS)	377

List of Figures

Figure 5-9. RSO Dialog Box	378
Figure 5-10. Revision Log Dialog Box (SOS)	380
Figure 5-11. Snapshot Objects Dialog Box	381
Figure 5-12. Tag Objects Dialog Box	382
Figure 5-13. Set Branch Dialog Box	384
Figure 5-14. Add Project Dialog Box (SVN)	386
Figure 5-15. Create Work Area Dialog Box (SVN)	387
Figure 5-16. Revision Log Dialog Box (SVN)	389
Figure 5-17. Discard Edits Prompt	392
Figure 5-18. Check in Edits Dialog Box	394
Figure 5-19. Manage Hierarchy Dialog Box	396
Figure 6-1. Import OpenAccess Library	408
Figure 6-2. Import OpenAccess Library Layouts Dialog Box	410
Figure 6-3. Export OpenAccess Symbols and Schematics Dialog Box	412
Figure 6-4. Export OpenAccess Layouts Dialog Box	413
Figure 8-1. Invoking the Language Editor Using the Popup Menu.	455
Figure 8-2. The Language Editor in Pyxis Schematic	456
Figure 8-3. Setup Preferences Dialog Box (Language Interface Panel)	457
Figure 8-4. Project Compilation Options Dialog Box	459
Figure 8-5. Compilation Options for <Verilog_file_name> Dialog Box	461
Figure 8-6. Compilation Options for <VHDL_file_name> Dialog Box	461
Figure 8-7. Project Registration Options Dialog Box	465
Figure 8-8. Symbol Layout Properties Dialog Box	469
Figure 8-9. Register Model from Source Dialog Box	472
Figure 8-10. Register Model from Library Dialog Box	475
Figure 8-11. Map Pins Dialog Box	478
Figure 8-12. Map Properties Dialog Box	481
Figure 8-13. Compiled Library Mappings Dialog Box	485
Figure 8-14. Import HDL Dialog Box	488
Figure 8-15. Compilation Options Dialog Box	489
Figure 8-16. Import Registration Options Dialog Box	490
Figure 8-17. Import Spice Dialog Box	491
Figure H-1. Parts of a Location Map	532
Figure H-2. Reading a Location Map Into Memory	539
Figure H-3. Location Map Resolves an Existing Design Reference	541
Figure H-4. Location Map Server Environment	548
Figure I-1. File > Import Pulldown Menu Options	551
Figure I-2. Import Design Kit to Technology Library Dialog Box	552
Figure I-3. Import ICstudio Project Dialog Box	555
Figure I-4. Import IC Studio Library Dialog Box	558
Figure I-5. Import Classic Data Dialog Box	561
Figure I-6. Import External Library Dialog Box	562
Figure I-7. Import Custom View Dialog Box	563

List of Tables

Table 2-1. File system object icons in Pyxis Project Manager	53
Table 2-2. Pyxis Project Manager Keyboard Navigation Keys	58
Table 2-3. Browse for Object Dialog Box Options	65
Table 2-4. Pyxis Project Manager Pathname Functions	71
Table 2-5. Example of Processing a Soft Prefix	81
Table 2-6. Tool Viewpoint Functions	106
Table 2-7. Pyxis Project Manager Session Window Functions	113
Table 3-1. Navigation Buttons	160
Table 3-2. Symbol meaning	219
Table 3-3. Model Types	295
Table 3-4. Pyxis Project Manager Shell Commands	305
Table 4-1. Project Navigator Toolbar Icon Description	317
Table 5-1. Object Status Descriptions	363
Table 5-2. Starting and Ending Object Status	365
Table 5-3. Revision Control Menu Items	372
Table 5-4. Troubleshooting	405
Table 7-1. Pyxis Project Manager Function Summary	417
Table 7-2. Interactive Function Summary	433
Table 7-3. Design Object Toolkit Function Summary	440
Table 7-4. Configuration Toolkit Function Summary	445
Table 7-5. Tool Viewpoint Function Summary	448
Table 7-6. iDM Interactive Function Summary	449
Table 7-7. iDM Toolkit Function Summary	451
Table 8-1. Setup Preferences Dialog Box (Language Interface Panel) Settings	458
Table 8-2. Project Registration Options	466
Table 8-3. Symbol Layout Properties Dialog Box Contents	470
Table 8-4. Map Pins Dialog Box Contents	479
Table 8-5. Map Properties Dialog Box Contents	481
Table B-1. Design Object Properties Added by the Pyxis Project Manager	507
Table D-1. Pyxis Project Manager Design Objects	517
Table E-1. Design Object Types	519
Table E-2. Pyxis Custom Design Flow Application Tool Icons	521
Table H-1. Workstation Location Maps	549

Chapter 1 Overview

Pyxis Project Manager helps you to manage your designs by not only providing a structured framework for creating and organizing your design, but also providing simple methods to copy and move designs, access previous versions of designs, and perform other similar data management tasks..

The basic unit upon which Pyxis Project Manager operates is called the design object. Design objects are simply design files and directories. Instead of treating these files and directories as separate objects, Pyxis Project Manager treats them as a single object, which is the design object. Regardless of the various files and directories that compose your application's design objects, Pyxis Project Manager handles them in the same way. For example, using the same Pyxis Project Manager command, you can copy a schematic, a physical layout, or a document.

A subset of the design management operations is available from within the EDA applications. These design management operations are provided by the Integrated Design Manager (iDM), which can be accessed from any window using the pull-down menu under **MGC > Design Management**. The iDM commands are also described in detail in "["Integrated Design Management \(iDM\)" on page 131](#).

The Pyxis Project Manager Tool	17
Pyxis Project Manager Windows	19

The Pyxis Project Manager Tool

Pyxis Project Manager helps you manage your designs by displaying each design object as a single, distinctive icon.

As a result, you do not need to keep track of details, such as which files and directories compose which design object. In addition, you can recognize the type of the object instantly, by viewing its icon.

[Figure 1-1](#) illustrates some of the features of the Pyxis Project Manager graphical user interface.

Figure 1-1. The Pyxis Project Manager Graphical Interface

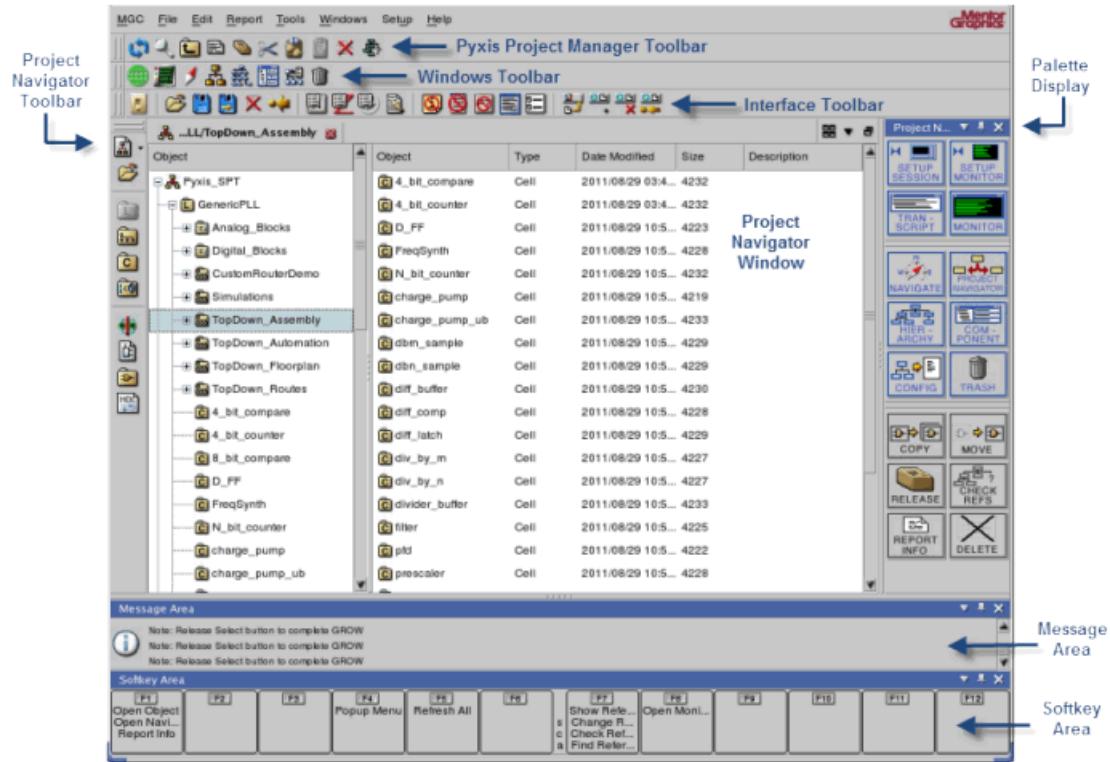


Figure 1-1 illustrates some of the features of the Pyxis Project Manager graphical user interface. Use the **Setup > Toolbars** menu to hide or display toolbars and the **Setup > Windows** menu to hide or display various windows such as the Message Area or Softkey area.

Design Navigation

Pyxis Project Manager allows you to navigate through your designs, using simple "point-and-click" mouse techniques. In Pyxis Project Manager, you can navigate through containers, which are basically *design objects*.

Data Centered or Direct Tool Invocation

Pyxis Project Manager's Tools Window displays all of the tools that you can invoke from your session. To invoke a tool in the Tools window, double-click the corresponding tool's icon with the Select mouse button. Alternatively, you can navigate to a design object from within Project Navigator and double click that design object's icon to open it in the default tool.

Design Data Configuration Management

Pyxis Project Manager provides commands to copy, move, rename, or delete design objects. In addition, the Check References command, not only checks the references of a design object, but

also enables the user to correct broken references. In addition, it allows you to preview the validity of the changes before applying them.

Pyxis Project Manager also provides a Configuration Window, with which you can build complex groupings of related design objects and operate on them as a single configuration. This allows you to copy or move the configuration while preserving all of the references. Another important operation that you can perform on a configuration is to release it, which essentially creates a protected copy.

Fully Scriptable Interface

All user interface operations are executed as AMPLE syntax commands, which are redirected to a transcript. These commands can then be replayed or integrated into a script in order to customize and automate design operation sets.

Register Third-party Tools and Data Formats

While Pyxis Project Manager fully supports all Pyxis Custom Design flow design tools, there may be third party tools that are required in order to complete a design. Pyxis Project Manager can be configured to recognize these additional design objects and launch their associated applications.

Related Topics

[A Design Object](#)

Pyxis Project Manager Windows

Pyxis Project Manager has several windows, and each window provides its own set of functionality that allows you to perform a set of related tasks.

The feature common to all the Pyxis Project Manager windows is that they help you to manage design objects or groups of design objects. All the Project Manager windows can be accessed from the **Windows** pulldown menu, except for the Information window, which can be accessed using the **Report** pulldown menu.

Session Window	20
Navigator Window	20
Tools and Toolbox Window	21
Project Navigator Window	22
Configuration Window	23
Monitor Window	23
Trash Window	24
Types Window	25
Information Window	26
The Change/ Fix References Window	26

Session Window

The Session window is the primary window in Pyxis Project Manager and provides access to all other Pyxis Project Manager windows. While there may be more than one instance of the other Pyxis Project Manager windows, there is only one session window.

Related Topics

[Pyxis Project Manager Windows](#)

Navigator Window

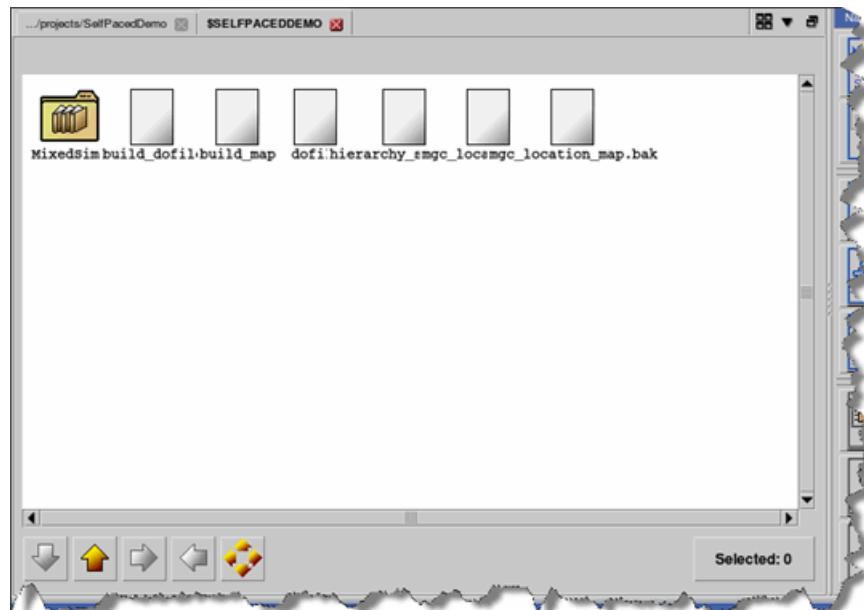
The navigator window displays the design objects of the current navigator directory.

The displayed objects can be filtered by their type or by a string pattern. In the navigator, you can select one or more design objects and choose an action to perform on it, including: opening the object, getting information about it, or changing its attributes.

If the selected object has references, you can explore the object's references as well as the references' parent containers. If the object is a container, you can explore its contents in the navigator window in any of the two viewing modes: **View by Icon** and **View by Name**.

[Figure 1-2](#) shows an example of the Navigator window in Pyxis Project Manager.

Figure 1-2. Navigator Window



Related Topics

[Pyxis Project Manager Windows](#)

Tools and Toolbox Window

The Tools window displays the tools found in various toolboxes that are currently available in your Pyxis Project Manager session.

You can switch between the Tools window and the Toolbox window using your right mouse bottom pop-up menu. The Toolbox window is used to displays the toolboxes that contain the tools, to add or remove toolboxes, or change the toolbox search path.

[Figure 1-3](#) shows an example of the Tools window, while [Figure 1-4](#) illustrates and example of the Toolboxes Window in Pyxis Project Manager.

Figure 1-3. Tools Window

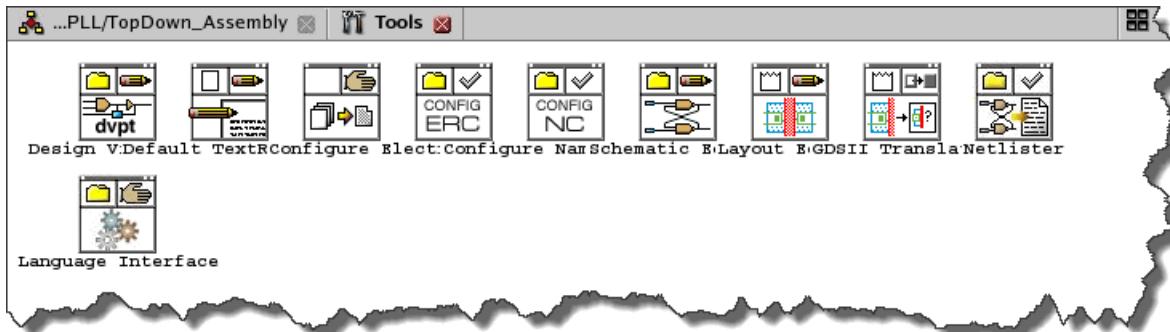
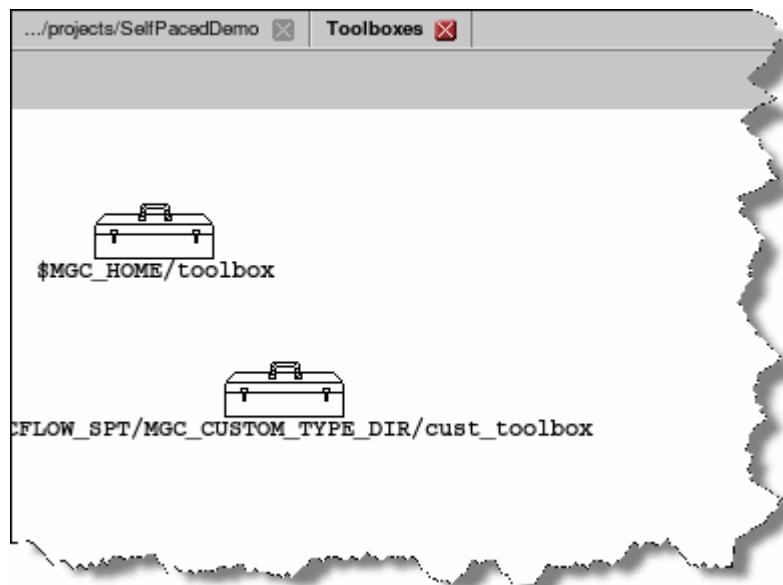


Figure 1-4. Toolboxes Window



Related Topics

[Pyxis Project Manager Windows](#)

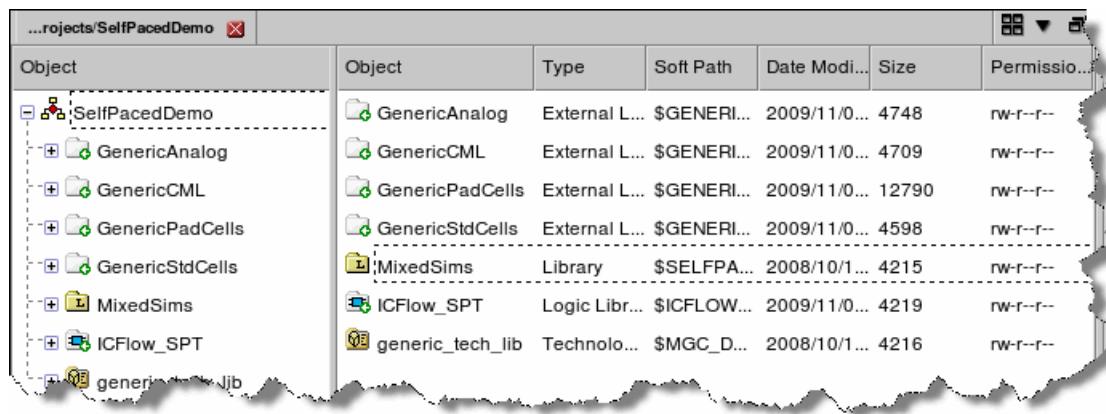
Project Navigator Window

The Project Navigator provides a structured framework for organizing and viewing IC design data within projects, libraries, cells and views, as well as an intuitive interface to create, access, and perform operations on hierarchical design data from one or more projects.

Projects contain internal design libraries, references to facilitate the access to external libraries, and a technology library with a specific technology configuration. The libraries can contain cells and their associated views.

Figure 1-5 shows an example of the Project Navigator window in Pyxis Project Manager.

Figure 1-5. Project Navigator Window



Object	Object	Type	Soft Path	Date Modif...	Size	Permissio...
SelfPacedDemo	GenericAnalog	External Lib...	\$GENERI...	2009/11/0...	4748	rw-r--r--
GenericAnalog	GenericCML	External Lib...	\$GENERI...	2009/11/0...	4709	rw-r--r--
GenericCML	GenericPadCells	External Lib...	\$GENERI...	2009/11/0...	12790	rw-r--r--
GenericPadCells	GenericStdCells	External Lib...	\$GENERI...	2009/11/0...	4598	rw-r--r--
GenericStdCells	MixedSims	Library	\$SELFPA...	2008/10/1...	4215	rw-r--r--
MixedSims	ICFlow_SPT	Logic Libr...	\$ICFLOW...	2009/11/0...	4219	rw-r--r--
ICFlow_SPT	generic_tech_lib	Technolo...	\$MGC_D...	2008/10/1...	4216	rw-r--r--
generic_tech_lib						

To create and manage projects using Project Navigator, a technology library needs to be created to contain all the reference design data necessary to create a design in an IC manufacturing technology.

The technology configuration specifically defines the userware path, the default process, and the rule files that are used for all the design objects created inside of the project. Each project and external library within the Project Navigator window is required to reference a single technology library with a specific technology configuration.

In any hierarchical design, there are references to a lower level cell from an upper level cell. If the lower level cell is renamed or relocated and the reference is not updated accordingly, it creates a broken reference. Project Navigator minimizes the occurrence of such broken references by automatically updating the references to a cell when they are moved or renamed.

Related Topics

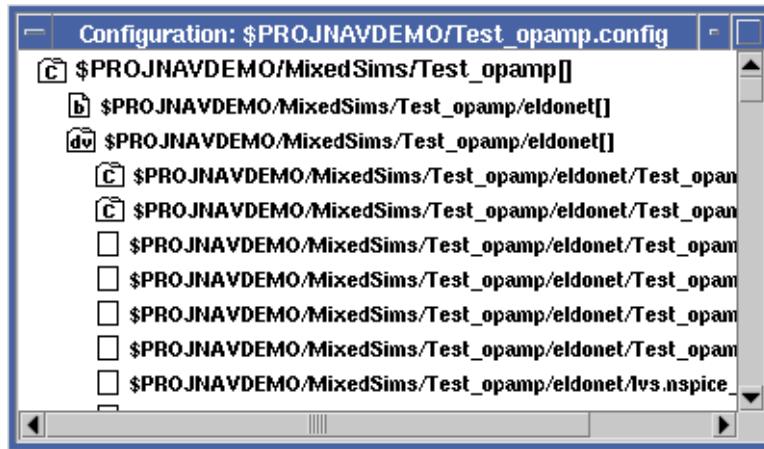
[Pyxis Project Manager Windows](#)

Configuration Window

The Configuration window is used for building hierarchical design configurations into a manageable object that you can copy and release.

[Figure 1-6](#) shows an example of the Configuration window in Pyxis Project Manager.

Figure 1-6. Configuration Window



Related Topics

[Pyxis Project Manager Windows](#)

Monitor Window

The Monitor window is a read-only area that contains status information about the last configuration operation performed. The Session Monitor window is used to view the real-time status reports of configuration management toolkit operations.

The Configuration Monitor window is also used to view the real-time status reports of interactive configuration management operations. Both the Session Monitor window and the Configuration Monitor window are usually referred to generically as the Monitor window.

[Figure 1-7](#) shows an example of the Session Monitor window, while [Figure 1-8](#) illustrates an example of the Configuration Monitor Window in Pyxis Project Manager.

Figure 1-7. Session Monitor Window

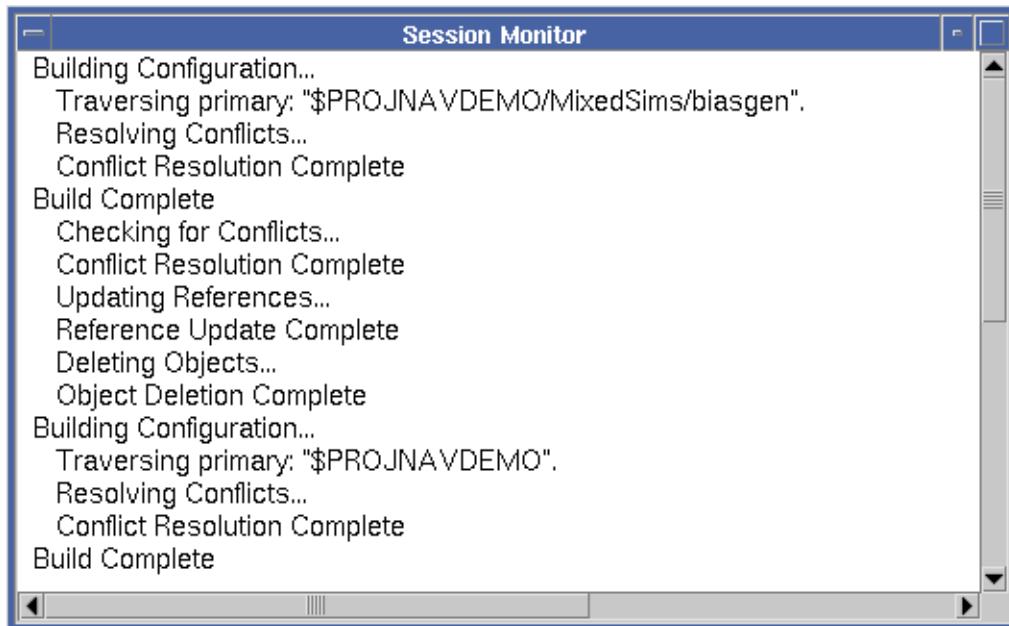
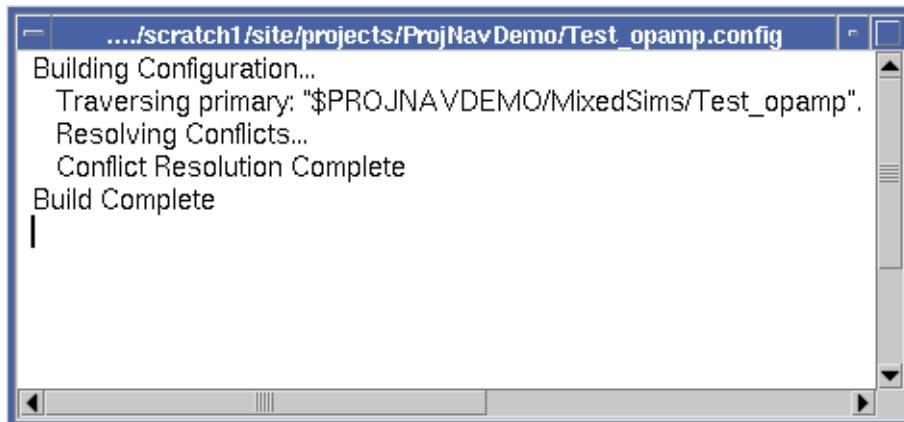


Figure 1-8. Configuration Monitor Window



Related Topics

[Pyxis Project Manager Windows](#)

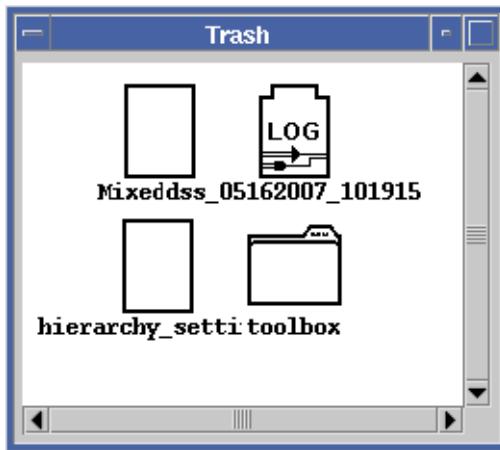
Trash Window

This window displays all the design objects that are dragged to the trash can.

You can either permanently delete one or more of these objects, or recover an object using the trash window's popup menu item Untrash Object.

[Figure 1-9](#) shows an example of the Trash window in Pyxis Project Manager.

Figure 1-9. Trash Window



Related Topics

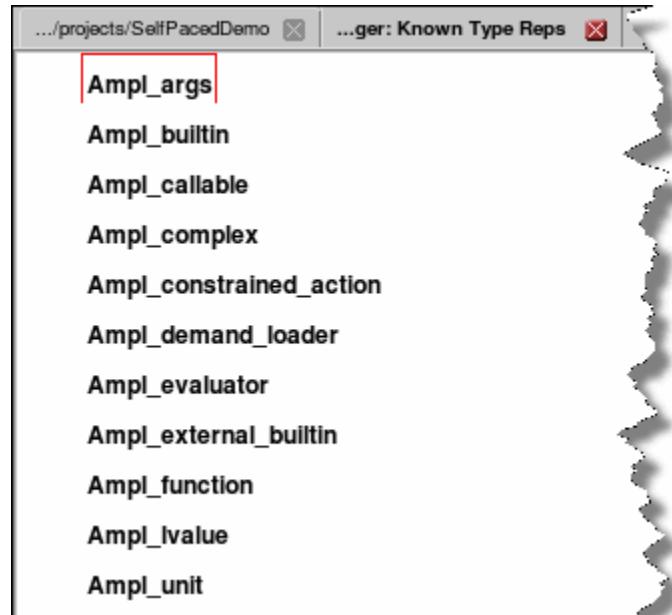
[Pyxis Project Manager Windows](#)

Types Window

This window displays a sorted list of all of the unique design object types known to Pyxis Project Manager.

Figure 1-10 shows an example of the Types window in Pyxis Project Manager.

Figure 1-10. Types Window



Related Topics

[Pyxis Project Manager Windows](#)

Information Window

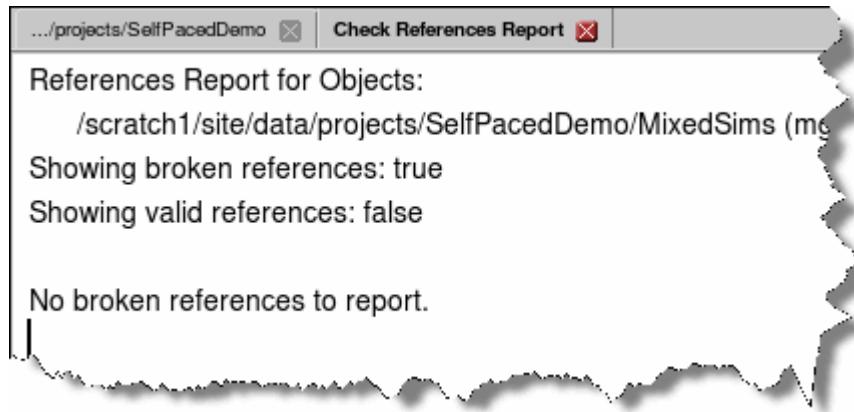
This is a read-only window that contains information about a selected design object.

An information window is displayed when you execute one of the Report > menu items. An information window title is of the format "XXXX Information Report", where "XXXX" is one of the following report types: object, tool, type, reference, or version.

When more than one information window of the same report type is opened, a number is concatenated to the end of the title in the form of #n, where 'n' is a positive integer. For example, when you execute the Report Tool Info menu item from the tools window popup menu, the Tool Information Report window is displayed. If you execute the Report Tool Info menu item again without closing the first information window, the new information window is entitled Tool Information Report#2.

[Figure 1-11](#) shows an example of the Information window in Pyxis Project Manager.

Figure 1-11. Information Window



Related Topics

[Pyxis Project Manager Windows](#)

The Change/ Fix References Window

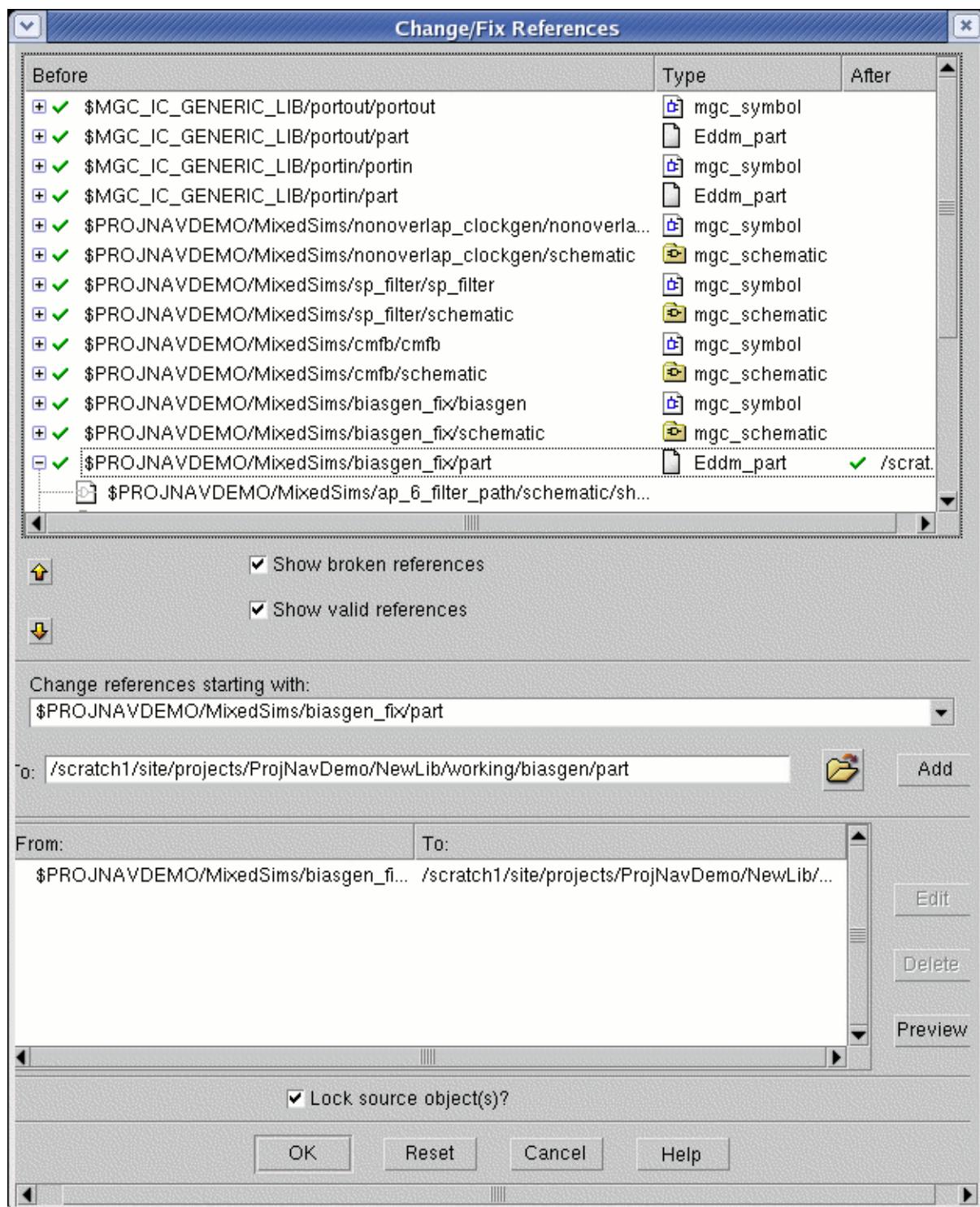
This window displays all the references held by the selected design object.

The references are color coded to indicate whether they are valid (green) or broken (red). Before one or more references are changed, the Change/Fix References window checks to make

sure that the new object is the correct type. It also provides a preview of the changed reference set before actually applying the changes.

[Figure 1-12](#) shows an example of the Change/Fix References window in Pyxis Project Manager.

Figure 1-12. Change/Fix References Window



Related Topics

[Pyxis Project Manager Windows](#)

Chapter 2

Key Concepts

To use Pyxis Project Manager effectively, there are some design management concepts, navigation and GUI tools as well as customization recommendations that you need to understand.

A Design Object	31
Pyxis Project Manager Display.....	54
Navigation.....	55
Pathnames in the Pyxis Project Manager	69
Design Management with Location Maps.....	71
Tool Invocation	102
Startup Files	112
Configuration Management Concepts.....	117
Integrated Design Management (iDM)	131

A Design Object

A *design object* is a collection of files and directories that models some aspects of a design.

In this discussion, *design* refers to the end result of any Mentor Graphics application. For example, you can think of a *component*, the output of Pyxis Schematic, as a design. The schematic, symbol, VHDL, and design viewpoint models which reside within the component are, in themselves, design objects. You need to be aware of design objects because they compose, and ultimately, represent your design. You also need to be aware of design objects because you must manage them; that is, you must move, copy, delete, and perform other such operations on them. The Pyxis Project Manager helps you perform these tasks.

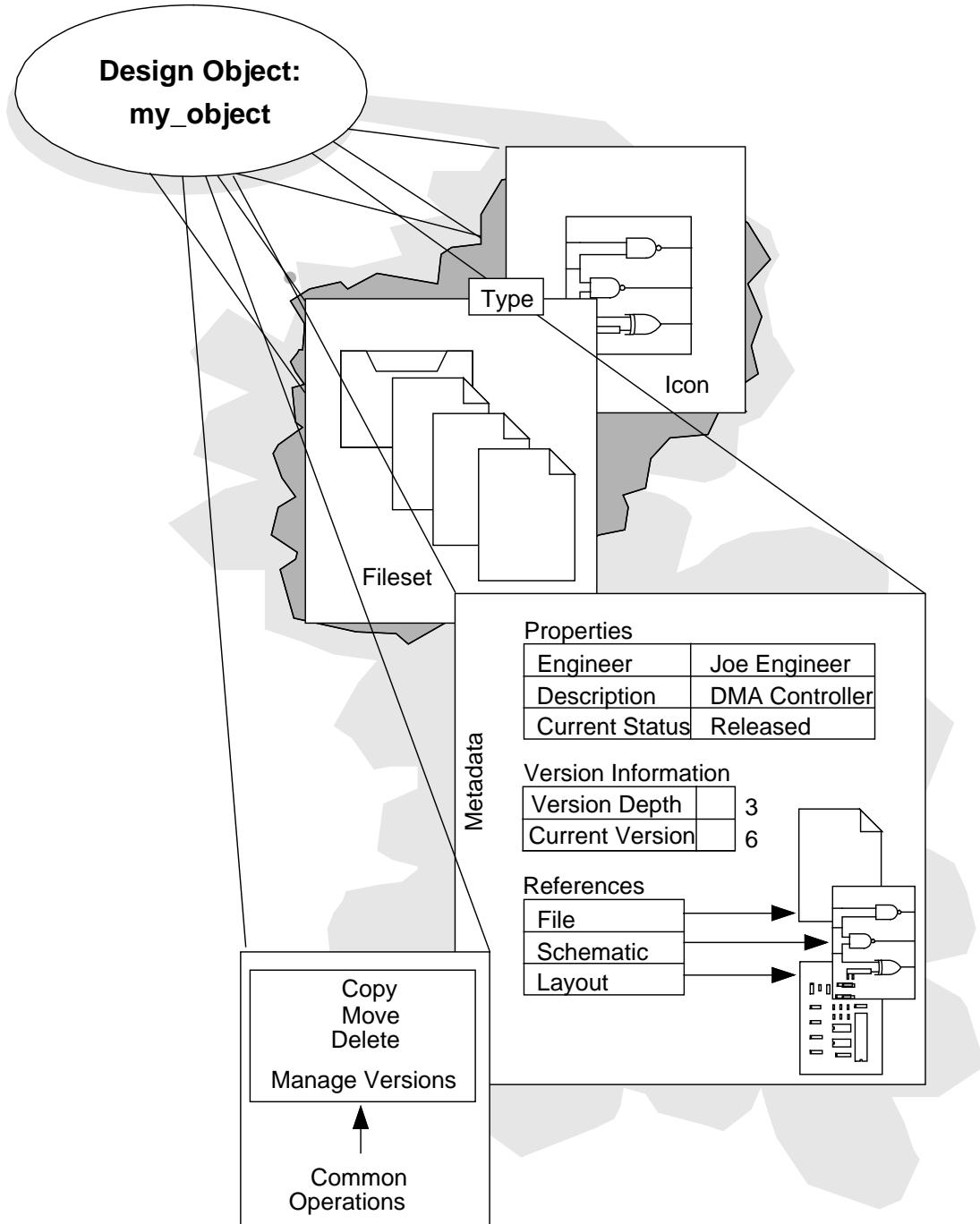
Each design object is composed of its own unique collection of files and directories. This collection is called the *fileset* of a design object. When you view a design object in the operating system shell, you see the files and directories that compose it. However, the Pyxis Project Manager and other Mentor Graphics applications view this collection of files and directories as a single object, a design object.

The particular configuration of each design object is very important; if one member of the design object's fileset is missing, your Mentor Graphics application may not be able to work with that design object. The main purpose of the Pyxis Project Manager is to relieve you of the task of manipulating the individual members of the design object's fileset. If you manipulate individual files in a design object's fileset with operating system commands, you make that

design object unusable. By using only the Pyxis Project Manager to manage your design objects between application sessions, you do not corrupt your design data.

Figure 2-1 shows the basic elements of a design object.

Figure 2-1. Basic Elements of a Design Object



The following list summarizes the basic elements of a design object:

- Name

Every design object has a name, and the name of each file in its fileset begins with that name. For each design object, the Pyxis Project Manager displays this name, along with an icon to indicate the object's type. Two design objects can share a name. That is, they can have the same pathname, but in this case, the objects must be of different *types*.

- Type

Each design object is an instance of a type. For example, a schematic design object could be an instance of type called *schematic*. The type holds information about that particular kind of design object, including what icon that the Pyxis Project Manager uses to display it, which tools can operate on it, and what files and directories compose the design object. In addition, from a design object's type, the Pyxis Project Manager knows whether a design object is versioned; that is, it knows if the design object can simultaneously contain several versions of the design data.

- Fileset

As stated previously, a design object is a collection of files and directories. These file system objects are a design object's fileset, and they each share the design object's name. In general, each fileset member's name has the following format:

name.suffix

where *name* is the name of the design object, and *suffix* is a string of characters that distinguishes one design file from another and often indicates the purpose of this fileset member. For example, the design viewpoint rules for the design object *my_obj* might be stored in a file called *my_obj.dvpt*. Each design object may include several files in its fileset, each with a different suffix to indicate its purpose.

- Metadata

Each design object contains design data, which resides in the design files within the object's fileset. In addition, most design objects contain *metadata*, which is data that describes the design object itself. In other words, metadata is information about the current state of the design object and its relationship with other design objects. Most metadata is contained in a special fileset called the *attribute file*. The attribute file contains the following items:

- References, which point to other design objects.
- Comment lines begin with a pound character (#) and list the host name and the hard pathname for the reference. The host name is the machine on which the attribute file was written; the host name has no connection to the name of the machine on which the referenced object resides. If the host name or hard pathname cannot be determined, the comment line for the corresponding field contains an empty string.

When the attribute file is read, these comment lines are ignored. Subsequent read and write operations may overwrite host name information that was previously there.

- Properties, which are arbitrary items of information about the design object. For example, a document design object might hold a property that indicates when the document was last edited.
- Version information, which is any information that pertains to the versions of the design object. A *version* is a numbered snapshot of the design object's design data at a previous time. Versions are numbered in the sequence in which they are created. Version information includes the number of the current version and how many versions the design object has.

In addition to the information held in the attribute file, the current state of the design object is indicated by the presence of other, temporary files in the fileset. For example, if a design object is locked (currently in use), its fileset contains a file with a *.lck* suffix.

- **Icon**

The Pyxis Project Manager navigators display each design object's fileset as a single, distinct icon. That icon visually indicates the design object's type.

- **Common Operations**

Using the Pyxis Project Manager, you can perform many operations on design objects, including copy, move, delete, and version management.

Filesets	34
Types	35
Versions	37
References	39
Properties	41
Common Operations on Design Objects	42
Special Design Objects	51

Filesets

A design object's fileset is the combination of files and directories that compose that design object.

The Pyxis Project Manager treats a fileset as a single object; when you use the Pyxis Project Manager to move, copy, or delete a design object, that object's entire fileset is moved, copied, or deleted.

In general, each fileset member's name has the following format:

name.suffix

where *name* is the name of the design object, and *suffix* is a string of characters that distinguishes one design file from another. A design object's fileset may have several files, each with the same prefix, but each with different suffixes.

For example, assume that you have a *schematic* design object called *fred* that is defined to consist of four files: *schem_id*, *.mgc_sheet.attr*, *.sgfx_1*, and *.ssht_1*. In the operating system shell, the fileset of design object *fred* would look like this:

```
schem_id
fred.ssht_1
fred.mgc_sheet.attr
fred.sgfx_1
```

In the Pyxis Project Manager, these four files appear as a single distinctive icon, with the name *fred* beneath or to the side of the icon. Although these files might coexist in the same directory with many other files and directories, the Pyxis Project Manager recognizes these four files as being a *schematic* design object.

Now assume that in the same directory, you have another schematic design object called *john*. In the shell, the fileset of design object *john* would look like this:

```
schem_id
john.ssht_1
john.mgc_sheet.attr
john.sgfx_1
```

To represent these four files, the Pyxis Project Manager uses the same icon as it used to represent the four *fred* files with the name *john* beneath the icon. Because they are both schematic design objects, they share the same icon in the Pyxis Project Manager.

Design object filesets are rarely as simple as this. Usually, they do contain an attribute file, and often they contain temporary files that indicate the current state of the design object. However, because the Pyxis Project Manager treats filesets as a single object, you usually do not need to know the details of a design object's fileset unless you are attempting to salvage a damaged design object.

Related Topics

[Design Object Filesets](#)

[A Design Object](#)

Types

Every design object is an instance of a type.

As stated previously, a design object's type holds information about that particular kind of design object. Some examples of design object types are *schematic* and *component*. You may have many *schematic* design objects in your design directories, and although each has a unique name and contains unique data, each can be opened by the same schematic editor. Each

schematic is a separate design object; that is, each of the schematic design objects is an instance of the type *schematic*.

A design object's type has the following characteristics:

- The combination of files and directories that composes the design object (the fileset). For example, the *schematic* type definition specifies that every schematic design object has files with the suffixes *.sgfx* and *.ssht*.
- Whether the design object is versioned or unversioned. If more than one version is allowed by the type, the object is said to be a “versioned” design object; otherwise it is “unversioned.”
- The default *version depth* for the design object, which is the number of versions that a design object keeps at any one time.
- The icon that Pyxis Project Manager navigators use to represent the design object's fileset. Note that a *schematic* and a *component* design object would have different icons. However, each *schematic* design object (each instance of the type *schematic*) has a unique name but the same icon.

Each type has a unique name. When managing your design objects, you may need to know that name.

In the Pyxis Project Manager, you can determine the name of an object's type by selecting the object and executing **Report > Object Info** from the navigator's popup menu. This menu path brings up a read-only window that displays information about the selected object, including its type.

You can also view a sorted list of known type names by executing the Open Types Window command in a popup command line, from any Pyxis Project Manager window. This command displays a read-only window that contains a sorted list of the design object types currently loaded in the Pyxis Project Manager. This list does not contain any duplicate entries; if more than one occurrence of a type exists, only the most recently added type is listed.

Type definitions are stored in a special file in the Mentor Graphics Tree which is located at *\$MGC_HOME*. This file is called a *type registry*. The Pyxis Project Manager understands these type registries and, as a result, can manipulate each type of design object found in the type registries. If you navigate to a set of files whose type is not stored in the Mentor Graphics Tree or specified via the *MGC_TYPE_REGISTRY* environment variable, the Pyxis Project Manager does not recognize those files as being a design object. As a result, the navigator only displays generic file and directory icons to represent each member of the fileset with the unrecognized type.

Appendix D contains a table that summarizes the design object types that are unique to the Pyxis Project Manager. To find out details about your application's design object types, refer to the user's manual for your application.

Related Topics

[A Design Object](#)

[Versions](#)

[Function Descriptions](#)

[Design Objects](#)

Versions

When you invoke an application and develop your design, you create design objects.

As you edit your design, those design objects change. The current and previous states of a data object are called *versions*. Mentor Graphics provides two kinds of design objects: *versioned* and *unversioned*. Versioned design objects retain previous versions of your design. Unversioned design objects retain only one version at any one time.

Note

 It is important to remember that containers may be versioned or unversioned design objects; however, a versioned container does not version its directories, only its attribute file metadata and other files.

Here are some of the operations that you can perform on versioned design objects in the Pyxis Project Manager:

- In the navigator, you can select an object and then execute **Report > Show Versions** from the navigator popup menu. Doing so brings up the *version window*, which displays the previous versions of the selected design object.

In the version window, you can select the versions and get information about them, invoke tools on them, and delete them. Because *version branches* are not currently supported, when you invoke tools on previous versions, you cannot edit them.

Also in the version window, you can copy a previous version to a new object name. Doing this creates a new design object and allows you to evolve it differently than the original.

- In the navigator, you can set the *version depth* of a design object, which is a number that specifies how many previous versions to keep at a given time. When you create a new version, old versions that are beyond the version depth are deleted. This is called *version pruning*. To keep more previous versions at a given time, you increase the version depth of the design object. The default version depth of a design object is specified by that object's type.
- You can *freeze* a specific version of a design object, which prevents deletion of the frozen version by version pruning. However, a frozen version is deleted if you delete the entire object. A frozen version does not apply to the version depth count and is retained until you *unfreeze* it. If you *unfreeze* a version that is older than the version depth, that

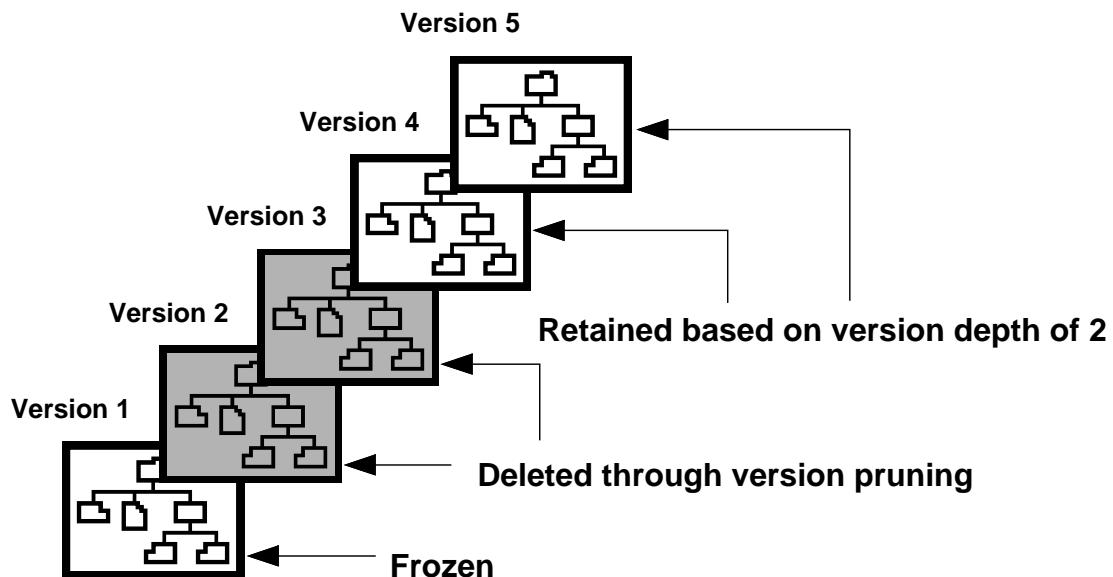
version is not immediately pruned. Instead, it survives until you either set the version depth or create a new version of the design object.

- You can make the previous version current by deleting the current version, using the Revert Version command. This is called *reverting* the version of an object.
- You can release a series of versions of a design object, to the same location multiple times; this results in a released object, in which each version corresponds to a particular release. However, if the object is unversioned, releasing multiple times to create a released object in which each version corresponds to a particular release is not possible.

When you release unversioned design objects, those unversioned design objects replace previously released instances of the object. When you release unversioned containers, not only the container, but all the objects it contains, whether versioned or not, are replaced. The exceptions to this rule are design objects of the type “mgc_container”; these containers are ordinary directories. Containers of this type are handled as a special case to allow designs contained in ordinary directories to layer during a release.

Figure 2-2 shows an example of a versioned design object with one version frozen. The design object has been written to disk five times. As a result, the current version is number 5. It also has a version depth of 2, and version number 1 is frozen. Figure 2-2 shows all five versions of the design object, but versions 2 and 3 have already been pruned and no longer exist in the file system. The next time that a new version is written, version 4 is deleted, leaving version 5 and a new version 6, as well as version 1, which is frozen. A frozen version does not apply to the version depth count and is retained until you unfreeze it.

Figure 2-2. Version Pruning and Freezing



Related Topics

[A Design Object](#)

[\\$\\$get_object_current_version\(\)](#)

[\\$\\$get_object_versions\(\)](#)

[Releasing a Configuration](#)

[\\$get_object_version\(\)](#)

References

A *reference* is a pointer from one design object to another.

Every design object can hold references to other objects, and each reference consists of a design object name (a pathname), a type, and a version specifier. Together, these three elements completely specify the referenced design object or a particular version of the design object.

Design objects use references to record a variety of relationships. For example, a schematic might reference a part in a library, or a component might reference a model. You can show the references of a design object, by selecting the design object in the navigator and executing **Report > Show References** from the navigator popup menu.

In addition to the references created by design tools, the Pyxis Project Manager allows you to add, change, and delete your own references, using either menu items or AMPLE functions. You might add a reference, for example, from your component to a specification document. A reference from your design to the documentation creates a useful configuration that, when released, contains the documentation as an integral part of the design.

Caution

 References define the structure of the entire design. To avoid corrupting your design data, use extreme caution when changing references.

Reference pathnames are kept in attribute files. Each object that contains references has an attribute file. All versions of referenced data are stored in a single attribute file. Attribute files are in ASCII format and, as a result, you can edit them within the file system. However, do not edit these files at the shell level. If you edit an attribute file directly, you can corrupt the design object to which it belongs. You should always use the Pyxis Project Manager to edit all your references.

Reference Types

Two types of references exist in the Pyxis Project Manager. The first type is a reference that relates to design objects. The second type is a reference that is created by another application. For example, the Pyxis Schematic application uses references to construct schematics from component libraries. References of the second type cannot be created in the Pyxis Project Manager.

Change or Delete References

In the Pyxis Project Manager, you can change or delete references that you create in the Pyxis Project Manager. However, with one exception, the Pyxis Project Manager prevents you from changing or deleting the references created by design tools. The one exception is the Pyxis Project Manager Change Object References command, which allows you to change a reference's target path and which you use to fix broken references after you move or delete a design object.

Copy or Move Objects

When you copy or move a design object or a configuration of design objects, the Pyxis Project Manager automatically updates the reference pathnames of all design objects in the selected set, to reflect the new target location; this includes the reference pathnames of design objects that are contained by the selected design objects. As a result, when copying or moving in the Pyxis Project Manager, you need not locate each item in the copied tree and change its references because automatic reference updating is performed. However, you only get automatic reference updating if you perform these operations while in the Pyxis Project Manager.

Move or Delete Objects

If you move or delete design objects, the references of other design objects that point to them become out of date or “broken.” That is, the references of other design objects that point to them still point to the previous location. For example, assume that the design object `$PROJ_DESIGN/d1/my_obj` references `$PROJ_DESIGN/d2/your_obj`. If you move `$PROJ_DESIGN/d2/your_obj` to `$PROJ_DESIGN/d1/your_obj`, the reference from `my_obj` to `your_obj` is broken. To repair the broken reference, you use the Change Object References command.

References can have their own set of properties, which allows design tools (or you) to record information about references.

Reference States

References have a *state*. A reference's state determines the mode in which the referenced object is accessed through the reference. References can have the following states:

- Current

A current reference always points to the current version of the referenced object. When you create a new version of the referenced object, a reference with this state automatically points to the newly created current version. The current state is the most common reference state.

- Read-only

A read-only reference is similar to a current reference. Like a current reference, a read-only reference points to the current version of the design object. However, you cannot

edit this object when you access it through the reference, regardless of the access permissions of the referenced object.

- Fixed

A fixed reference points to a particular version of a design object. A fixed references is always a read-only reference. You can only fix a reference while pointing to the current version of an object. Then, as the design object evolves, a fixed reference still points to the version to which it was fixed.

The most important use of reference states in the Pyxis Project Manager is in the configuration window, which allows you to gather large groups of design objects. To gather these design objects, the configuration window uses references that associate these design objects with each other. Depending on the state of these references, you can create configurations that track the current state of each design object or that reflect a snapshot of an entire design.

Reference Navigation

In addition to displaying references in the Pyxis Project Manager, you can navigate references. For information about fileset types, file extensions, or versions, refer to Appendix C.

Related Topics

[Function Descriptions](#)

[Copy Design Object](#)

[Move Design Object](#)

[Navigation](#)

[References](#)

[Specifying the Build Rules](#)

Properties

A *property* is information that applications can add to design objects, versions, and references.

A property consists of a name, which is a character string, and a value, which is also a character string. For example, a design object might have the following property:

“Last_Edited” “7 FEB 1999”

where “Last_Edited” is the name, and “7 FEB 1999” is the value of the property.

For another example, a reference from a component to a specification document might hold this property:

“Reference_Type” “Specification Document”

In this example, neither the component nor the document hold the property. Rather, the reference holds the property.

In addition to objects and references, previous versions of a design object can hold properties. These properties are visible only from the Pyxis Project Manager version window, and they do not propagate to the next version, as object properties do.

In the Pyxis Project Manager, you can add properties to design objects, references, and versions. In addition, you can edit or delete any object, reference, or version property that you can view in the Pyxis Project Manager.

The Pyxis Project Manager and other tools also assign their own tool-specific properties for internal use. An example of a tool-specific property is the “creating_tool” “project manager” property that the Pyxis Project Manager attaches to all references that you create with the Pyxis Project Manager Add Reference command.

Caution

 Tool-specific properties contain vital information about a design. To avoid corrupting your design, use extreme caution when editing these properties in the Pyxis Project Manager.

The presence of the “creating_tool” “project manager” property allows the Pyxis Project Manager to delete these references; it cannot delete references that do not hold this property. Remember, if you can view the object, reference, or version property in the Pyxis Project Manager, you can edit it in the Pyxis Project Manager.

Related Topics

A Design Object

Design Object Properties

Getting Information about a Version

Pyxis Project Manager Standard Properties

Common Operations on Design Objects

The Pyxis Project Manager provides all of the operations that you need to manage design objects.

These operations are all available in the navigator, which displays design objects as icons. The navigator is described in detail in the section on “[Navigation](#)” on page 55 in this chapter.

.Note

When you copy or move data that you are unfamiliar with to your workstation, before beginning work, you should check its references using the `$check_references()` command. Because the Pyxis Project Manager copies and moves data as it exists at the source, if the data at the source has broken references, the data at the destination also has broken references.

Display Information	43
Copy Design Object	44
Release Design Object	46
Move Design Object	48
Rename Design Object	50
Delete Design Object	50
Salvage Design Object	51

Open Design Object

You must open a design object to edit or display its contents.

When you open a design object, the Pyxis Project Manager determines the design object's type and performs an appropriate action. For example, when you open a directory, the Pyxis Project Manager navigates inside the directory and displays its contents. When you open a schematic design object, the Pyxis Project Manager invokes a schematic editor, using that design object as input.

In the navigator, you can open all design objects by selecting the design object and choosing **Open >** from the popup menu. If several tools can operate on this design object, the **Open >** menu item cascades to a list of compatible tools. For example, when you select a directory to open, the **Open >** popup menu cascades to the **Explore Contents** menu item. Because the list of compatible tools is obtained from the tools window, if a compatible tool is not in the tools window, the tool is not listed when the **Open >** menu item cascades.

You can also open *containers* and container design objects in the navigator by double-clicking on the container object. Double-clicking on an object causes the contents of the container to be displayed in the navigator window. If you double-click on a non-container object, an error messages displays stating that the selected object cannot be explored.

Related Topics

[Common Operations on Design Objects](#)

[Opening a Design Object](#)

Display Information

You can display information about the current state and nature of a design object by selecting the design object and executing **Report > Object Info**.

A read-only window appears, which displays important information about the design object, including the object's full pathname, type, properties, version depth, and fileset members.

Related Topics

[Common Operations on Design Objects](#)

[Getting Information about a Design Object](#)

Copy Design Object

You can copy a design object or a set of design objects to another location.

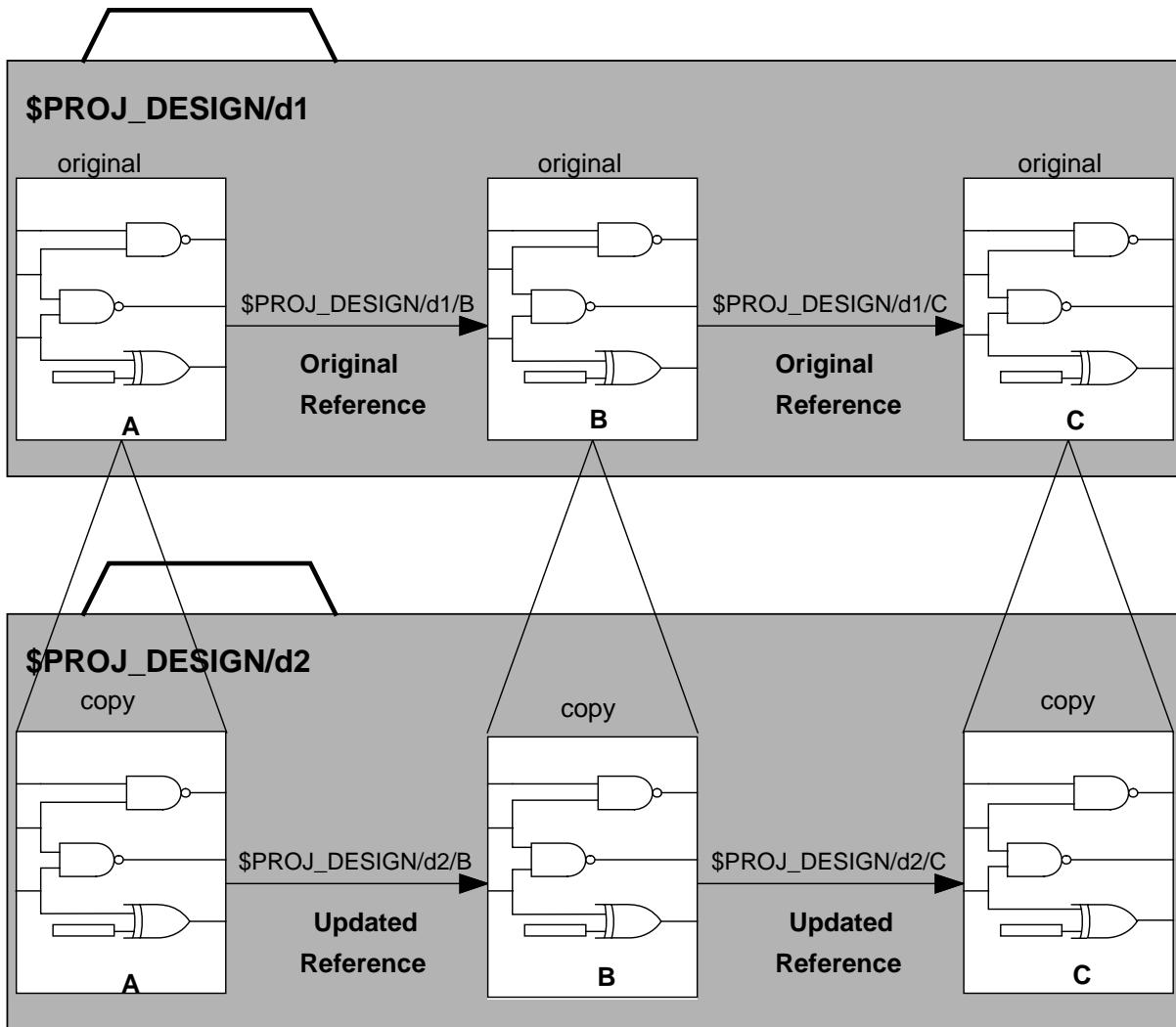
Copying leaves the source design objects intact and creates duplicates in the specified location. In a single copy operation, the Pyxis Project Manager copies each member of the source design object's fileset to the destination directory. In addition, all objects contained by the source object are also copied. For example, if you copy design object *A*, which contains *B*, and *B* contains *C*, all three objects are copied to the destination. In the destination design object, *A* contains a copy of *B*, and the copy of *B* contains a copy of *C*.

In the Pyxis Project Manager, the copying command has these special features:

- You can copy all versions or only a single version of the source design object. By default, copying from a navigator copies all versions of the source design object. To copy a particular version of a design object, you must show the versions of the design object by displaying a version window, selecting one of the versions, and executing the Copy Version command from the version window. At the destination directory, the copied version becomes version number 1 of a completely new design object.
- When you copy a set of design objects that have references to one or more of the objects within the set, the Pyxis Project Manager automatically updates those references to reflect the new location.
- You can specify that the copy operation traverse the design object's reference hierarchy and, thereby, include externally referenced objects in your target design object.
- You can filter out design objects from the copy operation based on the object's pathname.
- You can preview the results of the copy operation, before actually executing the operation.

Figure 2-3 shows reference updating during a simple copy operation. In this example, a set of design objects, *A*, *B*, and *C*, is copied from directory `$PROJ_DESIGN/d1` to directory `$PROJ_DESIGN/d2`. Note that in directory `$PROJ_DESIGN/d2`, the references of objects *A*, *B*, and *C* now point to locations in directory *d2*. The Pyxis Project Manager updates references for you automatically during copy and move operations. If you had not used the Pyxis Project Manager to copy these objects, their references would still point to locations in directory *d1* and you would have had to change the reference pathnames of each copied design object.

Figure 2-3. Reference Updating During a Copy



The Pyxis Project Manager updates the references *between* the objects in the selected set. If you select and copy A in one operation, B in a second operation, and C in a third operation, A, B, and C are not a single, selected set, and no reference updating occurs. As a result, \$PROJ_DESIGN/d2/A still points to \$PROJ_DESIGN/d1/B, and \$PROJ_DESIGN/d2/B still points to \$PROJ_DESIGN/d1/C.

If you do not copy A, B, and C simultaneously, the Pyxis Project Manager does not know where to point the references of A and B, so it leaves them unchanged.

Related Topics

[Common Operations on Design Objects](#)

[Copying Design Objects](#)

Release Design Object

You can release a design object or a set of design objects to another location.

A release creates a protected copy of a design object. The `$release_object()` command is a simplified interface to the configuration functionality provided with the Release Configuration command. The Release Object command allows you to release a set of design objects to a destination directory without setting build rules and building your configuration. The Release Object command has the following major features:

- The Release Object command is performed from the navigator window or from the toolkit. This command does not require that you specify build rules or build your configuration, prior to its execution.
- The Release Object command preserves containment relationships, updates references, and renames versions starting at number 1, in the target directory.
- If you do not specify a configuration name, the Release Object command releases a default configuration named “`release_config`”.
- If a location map is being used, this command also releases the current in-memory location map to the destination. The name of the location map object is of the format:

`configuration_name.cfg_map_cfg_version_num`

The `configuration_name` string is the specified name of the configuration, or the default name “`release_config`”. The `cfg_version_num` is the version of the configuration object in the destination directory.

- As part of the release operation, you can choose to have the references on the design object updated to reflect the new location.
- When you perform a simplified release on a selected set of design objects, the current version of each object is released to the destination directory, unless the object is referenced as a fixed version. If the object is referenced as a fixed version, the specified version is included in the release.
- The `$release_object()` command assigns the “released” property to every versioned object in the release destination directory. As a result, the Pyxis Project Manager does not allow you to create new versions of these released design objects. This is the protection that releasing your data provides.

If you want to open and edit the released design objects, you must copy the released object to another directory. The Copy Object command does not assign the “released” attribute.

Remember that the “released” attribute only prevents you from opening and editing *versioned* objects in the release directory. It does not prevent you from opening an unversioned design object, or from deleting or moving *any* released design objects. To preserve the integrity of your release, avoid these operations in the release directory.

- When you release the same design object or set of design objects to the same target directory multiple times, the Release Object command “layers” the versioned design objects, creating a new version of the object for each release.

For versioned design objects only, releasing does not overwrite old data with new data. If a versioned design object is unchanged since the last release, the Release Object command does not copy the object to the target directory. If the object has changed (a new version of it has been created), the Release Object command merges the new version with the existing design object in the target directory, thus creating a new version of that object.

Note

 Release layering only occurs for versioned design objects. The data in unversioned objects is overwritten with each repeated release to the same directory. When unversioned containers are released, not only the container, but all the objects it contains, are overwritten. The one exception to this rule applies to containers of type “mgc_container” which are ordinary directories. Containers of this type are handled as a special case, to allow designs contained in ordinary directories to layer during a release. With the exception of this special case, **containers should always be versioned** to allow their contents to be layered during the release process.

You need to ensure that important data are contained in “versioned” design objects, if you want to layer your releases. An alternative to layering is to choose a new directory for each release.

- The `$release_object()` command provides you with options for specifying reference traversal, and excluding objects from the release, based on their pathname.
- This command provides you with the option of previewing the expected results of the release operation in the session monitor window, prior to its actual execution.
- The `$release_object()` command also reports general status information and warnings about broken references, in the session monitor window. When the operation is complete, you can view the session monitor window by executing the pulldown menu item **Windows > Open Session Monitor**.

If your session setup values specify to show the monitor window, the session monitor window is opened when you execute this function.

Related Topics

[Common Operations on Design Objects](#)
[Change Your Session Setup](#)

[Releasing a Configuration](#)
[\\$release_object\(\)](#)

Move Design Object

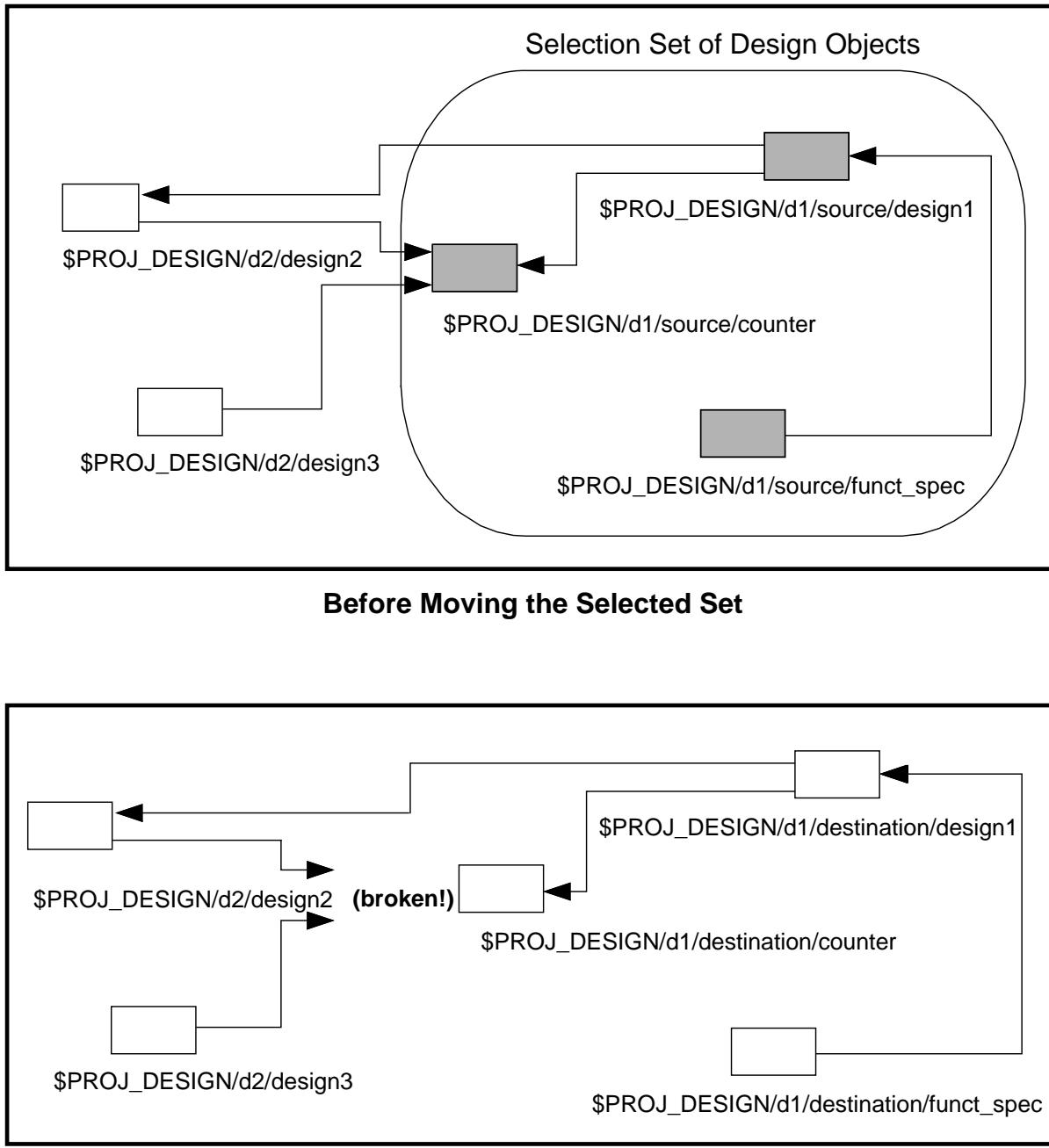
You can move a design object or a set of design objects to another location.

Unlike with copying, when you move a design object, all versions of the design object are moved. When you move a design object, as with copying, all objects in the selected object's containment hierarchy are also moved. Also, as with copying, when you move design objects that refer to other design objects in the selected set, the Pyxis Project Manager updates those references to reflect the new location.

Although the Pyxis Project Manager updates references between objects in the selected set, it does not update those that point to the selected set from outside the selected set. These references are broken as a result of moving an object or set of objects. To use these references after the move, you must manually update them with the Pyxis Project Manager Change Object References command.

Figure 2-4 shows reference updating and breaking during a simple move operation. In this example, the design objects `design1`, `counter`, and `funct_spec` are moved from `$PROJ_DESIGN/d1/source` to `$PROJ_DESIGN/d1/destination`. The references from `$PROJ_DESIGN/d2/design2` and `$PROJ_DESIGN/d2/design3` still point to `$PROJ_DESIGN/d1/source`. As a result, these references are broken.

Figure 2-4. Reference Updating and Breaking After a Move Command



Related Topics

[Common Operations on Design Objects](#)
[Moving Design Objects](#)

[Changing Reference Pathnames](#)

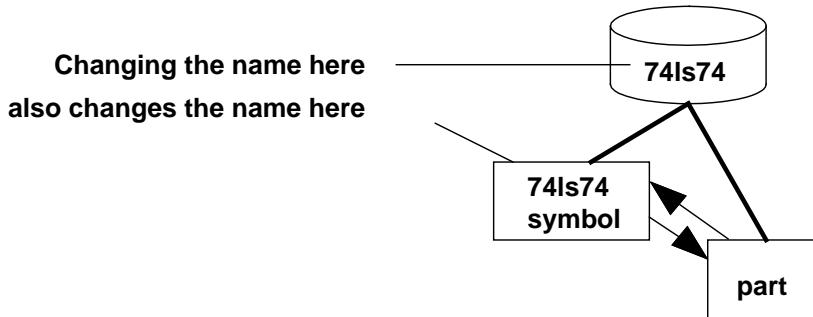
Rename Design Object

Renaming a container design object renames the design object and updates references within the containment hierarchy to reflect the new name.

External design object references that point to objects in the renamed container are not updated. Therefore, you must manually update these references.

When you change the name of a component that contains only one symbol with the same name as the component, this function sub-invokes the CIB to change the part interface so that the symbol is automatically renamed to match the component. For example, Figure 2-5 shows a component named 74ls74 that contains only one symbol, which is also named 74ls74. In this case, if you renamed the component to “foo,” the symbol would automatically be renamed to “foo” as well.

Figure 2-5. Renaming a Component Containing a Symbol



However, if there is more than one symbol, or if there is a single symbol but with a name that is different than the component, only the name of the component changes.

Related Topics

[Common Operations on Design Objects](#)

Delete Design Object

You can delete a design object or set of design objects in a single operation.

Deleting removes all versions, including frozen versions, of the selected objects. If a design object is a container, deleting the container also deletes all design objects that it contains.

In the Pyxis Project Manager, you can delete a design object in two ways:

- Delete with a command by entering the command on the popup command line.

When you delete with a command from the popup command line, the Pyxis Project Manager displays a dialog box that asks you to confirm the deletion. If you confirm the deletion, the selected design objects are deleted and you cannot retrieve them.

- Delete by dragging a design object to the trash. Until you empty the trash, you can open it and retrieve objects by dragging them out of it. You can only drag an object to the trash from an iconic navigator.

Related Topics

[Common Operations on Design Objects](#)

Salvage Design Object

You can salvage a design object that has been damaged by an application failure.

Design objects can become damaged in many different ways, but salvaging can repair only certain kinds of damage.

Salvaging restores design objects to a useful state after a catastrophic event, such as a power interruption, or hardware or software failure. If you are editing or writing a design object at the time, such an event often leaves files, that have been temporarily created in order to perform the edit or write, in the design object's fileset. As a result, the design object may be unusable. To understand how salvaging works, you need to know something about the temporary *lock*, *edit*, and *new edit* files.

Caution

 Do not use ordinary operating system commands to modify a design object's fileset unless you are explicitly instructed to do so in this manual as part of a salvage procedure. If you do use operating system commands without explicit instructions to do so, you can damage the design object in a way that the Salvage Object command cannot repair.

Any user who can edit the fileset of a damaged design object can salvage it. You should not attempt to salvage a design object at the same time another user is attempting to salvage the same object

Related Topics

[Common Operations on Design Objects](#)

[Metadata Files](#)

[Troubleshooting](#)

Special Design Objects

Some design objects provide special functionality.

Although these special design objects provide different functionality, the basic operations (described in the section “[Common Operations on Design Objects](#)” on page 42 can be performed on them.

Containers	52
File System Objects	52
Configuration Objects	53

Containers

Containers are design objects that can contain other design objects.

Only container objects can contain other objects, and they do so by having a directory as part of their fileset. However, containers are more than simple directories. A container's fileset can include other files and directories. Also, unlike directories, which the Pyxis Project Manager displays as generic, “folder” icons, containers have unique icons that indicate the purpose of the container.

Your environment may have several different “container” design object types, each with its own unique icon, fileset definition, and other unique traits. These container design objects share the common trait that they have a directory as a fileset member and thus can contain other design objects.

Note that containers are not versioned design objects.

Related Topics

[Special Design Objects](#)

File System Objects

A *file system object* may be a file or a directory.

In addition to recognizing the design objects that applications use, the Pyxis Project Manager also recognizes file system objects. Like all design objects, file system objects appear as icons in the navigator. However, unlike application-defined design objects, file system objects do not represent any aspect of a design.

File system objects are the Pyxis Project Manager interface to the file system. With them, the Pyxis Project Manager provides all of the basic file and directory management that the shell does. As in the shell, in the Pyxis Project Manager, you can create and edit directories and files. However, the Pyxis Project Manager surpasses the shell because it allows you to add properties and references to file system objects. File system objects are not versioned.

[Table 2-1](#) shows the icons that the Pyxis Project Manager tool uses in order to represent file system objects.

Table 2-1. File system object icons in Pyxis Project Manager

	
File	Directory

As with other types of design objects, when you open a file system object, the Pyxis Project Manager determines that object's type and performs the appropriate action. The following list describes the actions that may occur when you open a file system object:

- When you open a *file* a text editor invokes with that file as input. At any time, you can set up your Pyxis Project Manager to invoke any editor that you choose, including the Notepad editor; the next time you open a file, the editor you specified is invoked with that file as input.
- When you open a *directory*, the Pyxis Project Manager navigates into that directory and displays its contents. At any time, you can set navigator filters to specify that only a subset of the objects in the directory are displayed.

Related Topics

[Special Design Objects](#)

[Opening a File System Object](#)

[Specifying the Default File Editor](#)

[Pyxis Project Manager Design Object Types](#)

Configuration Objects

A configuration object contains a set of rules that allow you to define a collection of design objects and to treat them as one unit in the Pyxis Project Manager.

The design objects in this collection are called a *design configuration*, and they are interrelated by containment or references. A configuration object allows you to find all of the design objects that are associated with a design and operate on them as a single unit. These operations include copy, move, delete, release (protected copy) and lock.

The design objects in a design configuration may be related by very complex paths of containers and references. You might be able to recreate the relationship of these objects manually, by navigating and selecting, but the task would be difficult and lengthy; furthermore, you would have to do this before every Pyxis Project Manager operation. The configuration object automatically records all of the information needed to recreate the configuration instantly, at a later time.

Like all design objects, configuration objects are represented by a special icon. You can copy, move, and delete a configuration by simply selecting the unique icon which represents it. In addition, configuration objects are versioned. As a result, the configuration object can evolve with the design that it describes. Figure 2-6 shows the large and small icons for the configuration object. Large configuration icons represent configurations in an iconic navigator. Small configuration icons represent configurations in a list navigator.

Figure 2-6. Large and Small Configuration Object Icons



The process of creating and manipulating design configurations is called *design data configuration management*.

Related Topics

[Special Design Objects](#)

[Configuration Management Concepts](#)

Pyxis Project Manager Display

The Pyxis Project Manager work area is a window on your display called the *session window*.

With the mouse, you can interact with different areas of the session window. Most window areas are common to all applications.

The Session Window

The Pyxis Project Manager session window is the first window that appears when you begin your Pyxis Project Manager session. While you work in the Pyxis Project Manager, this window remains as the outermost bordered window. Figure 2-7 shows the Pyxis Project Manager session window. You can invoke the navigator, tools window, trash window, and configuration window from within the Pyxis Project Manager session window. Many windows may appear inside the Pyxis Project Manager session window, but only one window is active at a time. To perform an operation in a window, that window must be active. To activate a window, you move the mouse pointer into the window and press any mouse button.

Message Area, Dialog Boxes, and Prompt Bars

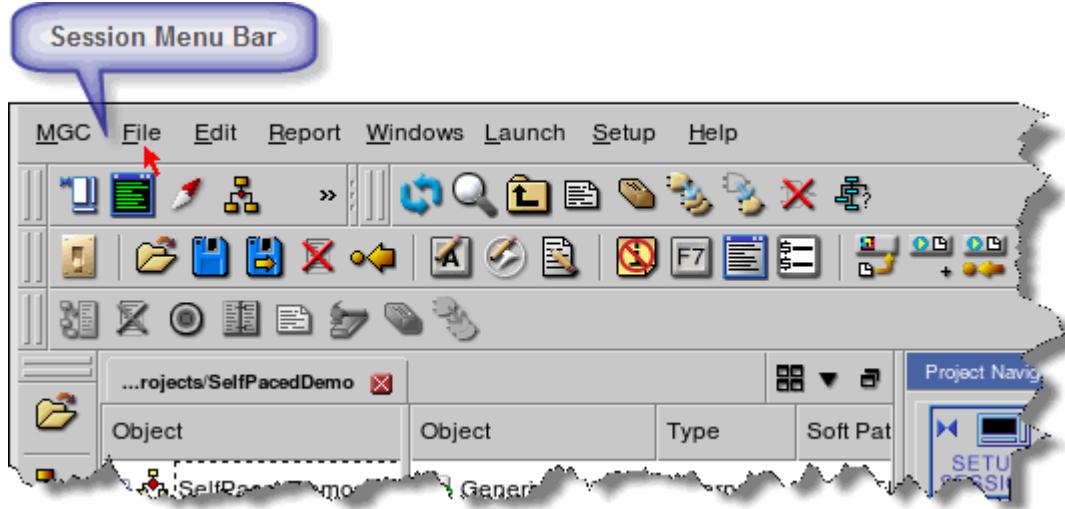
The Pyxis Project Manager reports information and errors by using the message area or a popup dialog box. The message area is a scalable line located at the bottom of the session window. A dialog box can appear anywhere within the session window and requests a response from you before it disappears. The actual position of the dialog box depends on the actions occurring at the time the message appears. The Pyxis Project Manager also uses dialog boxes and prompt bars to gather required input values from you. You may be prompted for input via a dialog box

or prompt bar, when you invoke a design tool. For example, many tools gather needed startup information before they run by first bringing up a dialog box.

Pulldown Menu Bar

The session window menu bar, at the top of the session window, provides the pulldown menus for a Pyxis Project Manager session, as shown in Figure 2-7. The menu bar selections change as you activate different windows.

Figure 2-7. Pyxis Project Manager Session Window



Related Topics

[Navigation](#)

Navigation

Navigation, along with tool invocation and design data configuration management, is one of the most important features of the Pyxis Project Manager.

Navigator.....	55
Navigator Components	59
Reference Navigation	62
The Object Browser Dialog Box.....	63

Navigator

The navigator is the most commonly used of the Pyxis Project Manager windows.

It provides access to design objects which are the basic unit of data in the Mentor Graphics design management system.

The Pyxis Project Manager navigator displays design objects. Instead of displaying the individual files of a design object's fileset, the navigator displays the object's name and, with a distinctive icon, the object's type.

Note

 If the navigator does not display a design object that you know exists in the current navigator directory and you have not specified any navigator filters that exclude that design object from the navigator window, the missing design object may have a filename extension that renders it "invisible" to the Pyxis Project Manager. The invisible file was created to manage Mentor Graphics data. For a complete description of the "invisible file" and its purpose, refer to "Invisible File" in the *Pyxis Registrar User's and Reference Manual* on SupportNet.

The navigator provides you with a dynamic filtering capability that allows you to filter which design objects are displayed in the navigator. Using UNIX System V wild cards, you can specify which design objects you see in the navigator, based on the design object's name and the design object's type. By default, navigator filters are set to include "*" which means that all files, except those files that begin with a ".", are displayed.

Viewing Modes

The navigator has two viewing modes: view by name and view by icon. Both modes display the same information: a group of design object icons and names. However, they use different formats:

- **View by Icon**

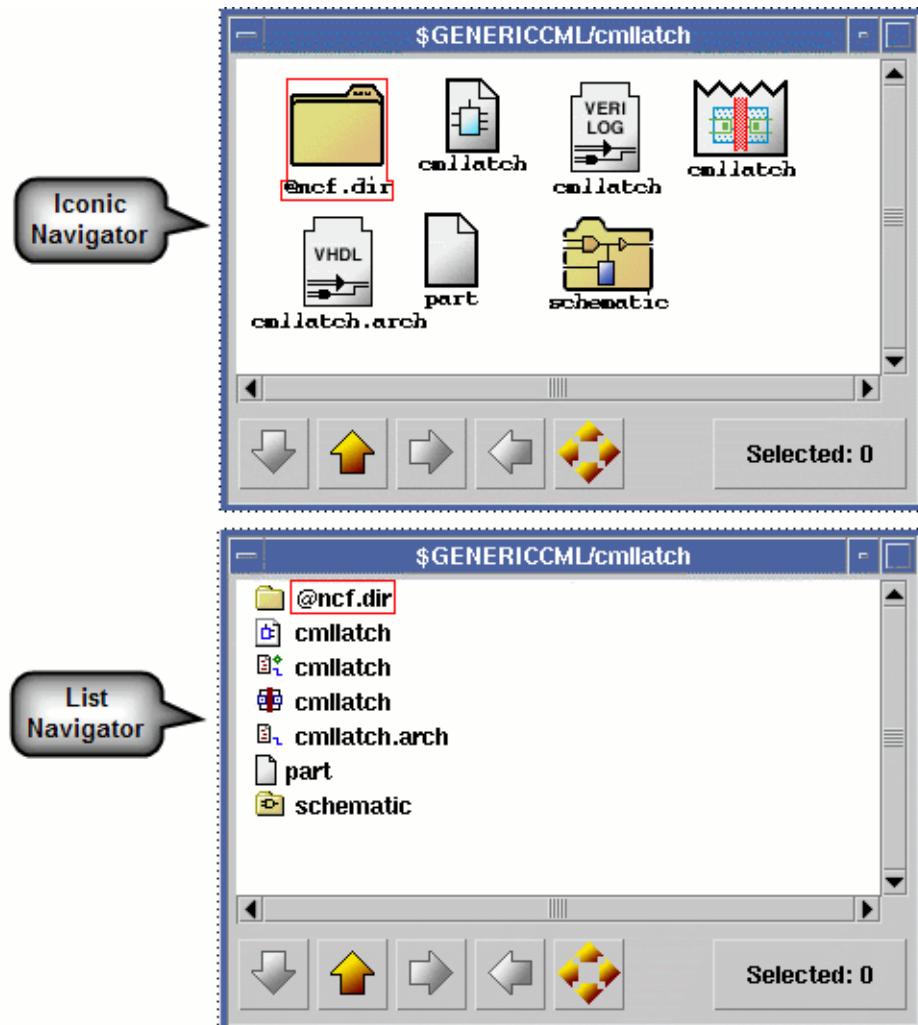
This mode displays an alphabetically sorted matrix of icons, with the design object name beneath the icon. The icons used are larger and more detailed than those found when you view by name. In this manual, a navigator set to view by icon is called an *iconic* navigator.

- **View by Name**

This mode displays a vertical list of icons, with the design object name to the right of the icon. The list is sorted alphabetically by design object name. The icons used are smaller and less detailed than those found when you view by icon. This navigator has the advantage of displaying more design objects in a given area. In this manual, a navigator set to view by name is called a *list* navigator.

Figure 2-8 shows two navigators: one in iconic mode, and one in list mode. The navigator window, in either iconic or list mode, supports both graphical (mouse) navigation and keyboard navigation.

Figure 2-8. Iconic and List Navigators



Keyboard Navigation

Keyboard navigation is based on the two primary highlight areas of the navigator window, as shown in Figure 2-8: the contained area and the button area. These two areas contain many of the navigator's components.

The contained area is either the list or iconic display of design objects. The button area is composed of the five arrow buttons beneath the contained area. If the contained area is the primary highlight area, a red box highlights the area, as seen in the list navigator in Figure 2-8. When the contained area is highlighted, the scroll keys can be used to move a sub-highlight over individual design objects within the contained area.

When the navigator window is initialized, the contained area is highlighted. You use the keyboard arrow keys to change the primary highlight area to the button area.

Table 2-2 lists the keys you can use to navigate in and between the contained and button areas.

Table 2-2. Pyxis Project Manager Keyboard Navigation Keys

Logical Key Name	Scope	Function
Up	Navigator contained area	Moves the sub-highlight to the previous entry.
	Navigator button area	Makes the contained area the primary highlight.
Down	Navigator contained area	Moves the sub-highlight to the next entry.
	Navigator button area	Makes the contained area the primary highlight.
Left	Navigator contained area	Makes the button area the primary highlight area.
	Navigator button area	Makes the next button the sub-highlight button.
Right	Navigator contained area	Makes the button area the primary highlight area.
	Navigator button area	Makes the previous button the sub-highlight button.
Enter	Navigator contained area	Selects the design object with the sub-highlight.
	Navigator button area	Invokes the sub-highlight button.
Shift-Enter	Navigator contained area (List navigator only.)	Selects all entries between the marked entry and the current sub-highlighted entry, including the sub-highlighted entry.
Ctrl-Enter	Navigator contained area	Toggles the selection of the sub-highlighted entry.

Related Topics

[Specifying Navigator Filters](#)

[\\$setup_filter_active\(\)](#)

[\\$setup_filter_all\(\)](#)

[Pyxis Project Manager's Graphical Interface](#)

[Navigator Components](#)

[Pyxis Project Manager Reference Manual](#)

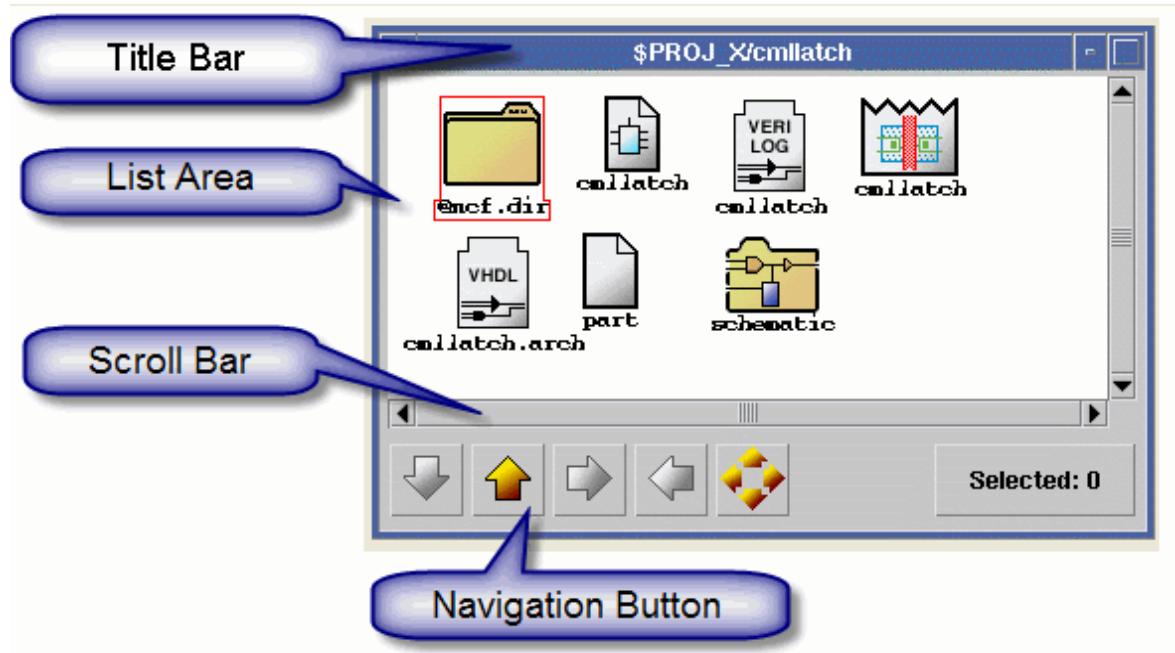
[Pyxis Common User Interface User's Manual](#)

Navigator Components

Displays by default.

The Pyxis Project Manager navigator GUI has several key components.

Figure 2-9. Navigator Components



Fields

Table 2-10. Navigator Component Options

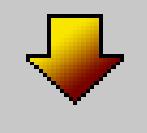
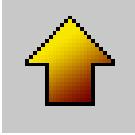
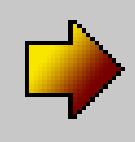
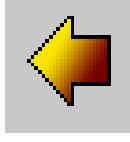
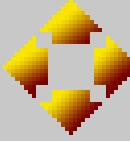
Field	Description
Title Bar	Indicates the current directory displayed in the window. If you are using a location map, the soft pathname of the current directory is shown. In Figure 2-9, the title bar displays the soft prefix “\$PROJ_X” which maps to a real location based on the entries in the location map. For example, “\$PROJ_X” could be the soft prefix which represents the project directory for the user's working group.
List Area	<p>Displays, in the list navigator a list of objects with an icon preceding each design object name. Displays, in the iconic navigator, a flat matrix of larger icons. In both the list navigator and iconic navigator, you can set navigator filters that determine which objects are displayed in the list area and the order in which they are displayed. If you do not specify any filters, all objects in the current directory are displayed in alphabetical order.</p> <p>The design objects in the list area are displayed according to one of the following selection criterion:</p> <ul style="list-style-type: none"> • The contents of the directory, which is specified in the title bar. Note that in this discussion, “directory” also refers to “container,” which is a special design object whose fileset includes a directory. • The references of the design object specified in the title bar. References are pointers from one design object to another. Unlike the operating system, the navigator allows you to navigate references.
Scroll Bars	Allows you to scroll the list or iconic area vertically and horizontally.
Softkeys	Provides you with information about which function keys are active for the active window and the commands to which they map. By default, the soft keys are visible. You can use the Setup > Session Defaults pulldown menu item to hide the softkeys.
Navigation Buttons	Allows you to navigate through the file system or to navigate references.
	<p>Explore Contents. Navigates into the selected directory and displays the design objects that are contained in that directory.</p> <p>When the contents of a directory are displayed in the navigator, you are in <i>contents mode</i>. You are always in contents mode, unless you have just explored the references of a design object. After you navigate into the selected directory, the navigator adds that directory's name to the title bar. If you have set navigator filters to exclude objects from the navigator display, those objects that meet the filter requirements for exclusion are not visible when you explore the contents of a directory.</p>

Table 2-10. Navigator Component Options

	<p>Explore Parent. If you are in <i>contents mode</i>, this button explores the container that contains the directory displayed in the navigator title bar. That is, this button navigates to the parent of the current design object's parent, and displays its contents. If you are in <i>reference mode</i>, this button explores the container of the selected reference's target object. That is, this button navigates to the parent of the object that is selected. After exploring the parent, the title bar displays the new location.</p> <p>Clicking the Explore Parent button in contents mode, executes the <code>\$explore_parent()</code> function. Clicking the Explore Parent button in reference mode, executes the <code>\$explore_reference_parent()</code> function.</p>
	<p>Explore References. Replaces the current display with the references of the selected design object. Reference pathnames are displayed exactly as they are stored in the attribute file. Exploring the references of a design object puts you in <i>reference mode</i>.</p> <p>When you explore a design object's references, the navigator adds that object's name to the title bar and follows the name with a “@”, indicating reference navigation. After exploring a design object's references, you can select another object in the display and explore its references or contents. When you alternate between exploring contents and references, the title bar reflects this by displaying both “/” and “@” in the pathname, signifying either a content or reference exploration.</p>
	<p>Explore Back to Parent. Navigates back to the design object that holds the references currently displayed. That is, this button navigates back to the design object from which you originally explored references. This button is only activated when you are in reference mode; that is, when you have selected an object and explored its references. When you are in reference mode, an “@” is the right-most character in the title bar.</p> <p>Clicking the Explore Back to Parent button in reference mode, executes the <code>\$explore_parent()</code> function.</p>
	<p>Go To. Displays a dialog box into which you can enter a soft or hard pathname to represent the file system destination to which you want to navigate. This destination must be a directory. After you enter the pathname of the destination directory and execute the dialog box, the navigator goes to that directory and displays its contents or the contents specified by your navigator filters, and the title bar displays the new location.</p>

Related Topics

[Project Navigator Graphical Interface](#)

Reference Navigation

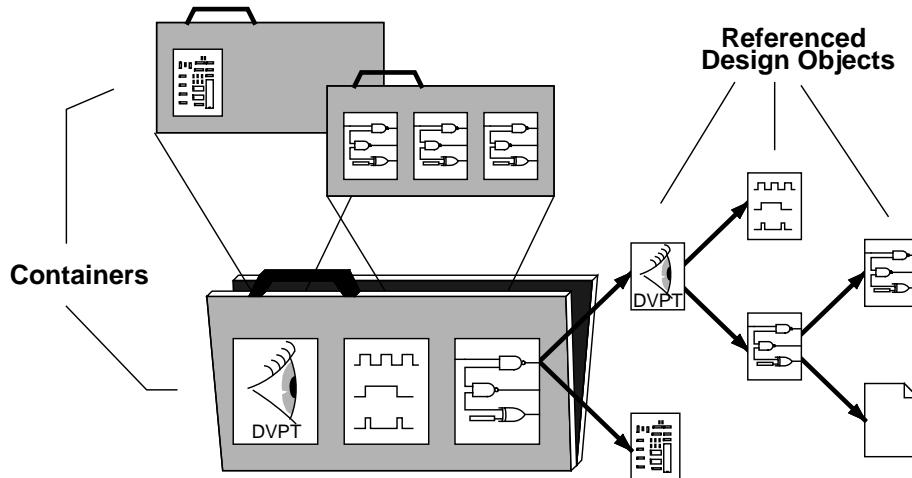
In addition to displaying icons instead of individual files, the Pyxis Project Manager navigator provides another powerful feature that the operating system shell does not provide: reference navigation.

This is a powerful alternative to simple file system navigation. Because references are arbitrary pointers, the design objects to which they point can be distributed throughout the file system. In the shell, if you want to operate on a referenced design object, you must list the references of the object that holds the reference, determine that it is the correct type, and then change your working directory to find it. To operate on multiple references, you must repeat this process for each referenced object.

In the Pyxis Project Manager navigator, you simply select a design object and explore its references by clicking the **Explore References** button. All of the referenced objects are displayed, and you can select and operate on any of them. Reference pathnames are displayed exactly as they are stored in the attribute file.

Figure 2-11 illustrates the exploration hierarchy by showing a container object holding both non-container objects and containers.

Figure 2-11. Exploring the Design



When you explore a container or explore references, the navigator's title bar uses container and reference path punctuation to reflect the location where you have moved. For example, the path `$MGC_HOME/pkgs/bin/usertool/XYZ` shows the containment traversal to the object named `XYZ`. In another example, if you explore the references of `XYZ/objectA`, which displays `objectB`,

and then you explore the references of *objectB*, the title bar displays the path *...XYZ/objectA@objectB@*.

Related Topics

[Project Navigator Graphical Interface](#)

The Object Browser Dialog Box

To access: **File > Object Browser**.

The *object browser dialog box* is a control that allows you to navigate your designs and to select design objects on which to work.

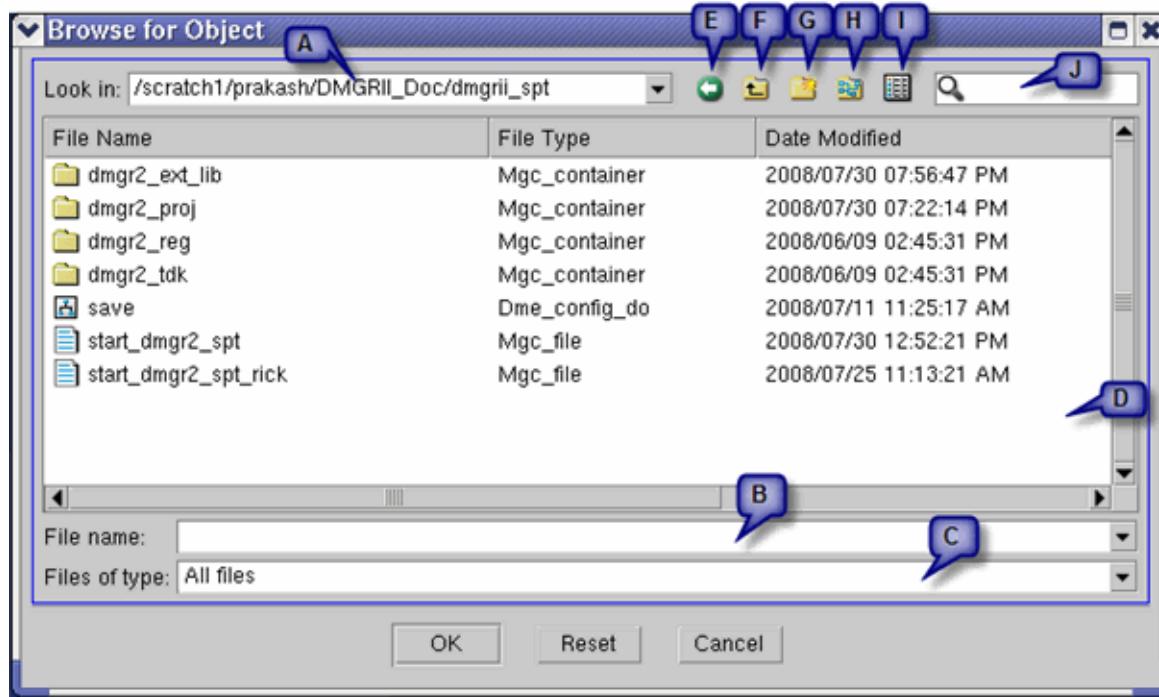
Description

Occasionally, your application might need to locate a design object on which to perform a task.

As an example, the Open Configuration command needs a configuration object to open, and the Copy Object command needs a design object to copy. If your application features the object browser dialog, you do not need to know the pathname to the configuration that you want to open, or the pathname to the object you want to copy. Instead, you can use the object browser dialog to find and to select the desired design object.

When you execute the dialog box in which the object browser dialog resides, the name and type of the design object that you selected is passed to your application.

Figure 2-12. Browse for Object Dialog Box



- A. **"Look In" field:** This field shows the current path for the objects being displayed in the browser. This field can also be hand-edited to browse to a specific location on the disk.
- B. **"File name" field:** This field allows you to type in the name of the file that you would like to select.
- C. **"Files of type" field:** This field contains the active file type used as a filter by the navigator.
- D. **View pane:** This is the main viewing area of the navigator. It displays a list of selectable objects, which are representations of files on the disk.
- E. **Back Directory button:** This button will move to the previous active directory.
- F. **Up Directory button:** This button will move to the parent of the current directory.
- G. **New Directory button:** This button will create a new directory within the current directory.
- H. **Browse Location Map Button:** This button will activate the "Location Map" mode, within which you will be able to browse your location map directories.
- I. **Change View Button:** This button allows you to change the current view mode.
- J. **Search Field:** This field will filter the view pane based on a simple text match between the view objects and the contents of the field.

Fields

Table 2-3. Browse for Object Dialog Box Options

Field	Description
Look in field	The look-in field displays the current path for the objects being displayed in the browser. You can also hand-edit this field to select a path of your choice, which in turn is activated when you press the 'Return' key. Alternatively, you can select a path from the drop-down list using the arrow button to the right of the text entry field. If the chosen path does not exist, the field automatically reverts itself to the current navigator directory. When in 'Location Map' mode, this field displays 'Location Map' in its text entry box. This field supports soft paths.
File Name Field	The file name field is provided so that you can specify exactly what you want selected in the navigator. If there is anything typed in the file name field, the navigator ignores whatever object is selected in the view pane and simply return whatever text has been typed into the field. The file name field automatically updates whenever an object is selected in the view pane. If you click the drop-down arrow to the right of the text entry box, a list of potential file names is displayed. This field is disabled if the navigator is not in single-select mode.
Auto Complete	When you begin to type in the file name field, the navigator looks for the best match given the characters entered, and auto-fill that match into the text field as highlighted text. For example, if there are files 'foo', 'bar', and 'baz' in a directory, and you type the 'f' character into the file name field, then you see the 'f' that you typed followed by a highlighted 'oo'.
Auto Filter	When you begin to type in the file name field, the navigator compiles a list of all potential matches and show a drop-down list containing those matches. For example, if there are files 'foo', 'bar', and 'baz' in a directory, and you type the 'b' character into the file name field, you see a drop-down box pop up with the entries 'bar' and 'baz'.

Table 2-3. Browse for Object Dialog Box Options

Field	Description
Filter Field	<p>The filter field is a method by which you can specify which kind of files you want to see. Each navigator has the 'All files' filter, followed by an optional subset of filters pertaining to the type that you are navigating for.</p> <p>For example, an 'Open Hierarchy' navigator could have a filter setup of 'All files', 'mgc_project', 'mgc_ext_lib', and 'mgc_tech_lib', with mgc_project being the default filter, since it is probably the most common hierarchy. When a filter is set to 'All files', all objects in the current directory are shown, regardless of type.</p> <p>When a type filter is chosen, the navigator displays all container objects (i.e. objects that can have other types of objects inside of them) plus all files that are of the specified type. It is possible to have filters with more than one type: for example, components and schematics can be combined into a single type. If the user clicks anywhere in the filter field, a drop-down list of the possible filters for the navigator appears. This field cannot be hand edited.</p>

Table 2-3. Browse for Object Dialog Box Options

Field	Description
View pane	<p>The view pane is a visual representation of the current navigator directory. There are two selectable views: List and Details.</p> <p>When a non-container object is double clicked in the view, this object is set as the navigator's value, and if the navigator is inside of a form, it closes.</p> <p>If a container object is double clicked, and the container object is NOT contained within the active filter, the navigator browses into that container, and it becomes the active directory.</p> <p>If the container object happens to be contained within the filter list, it behaves exactly as a non-container object would.</p> <p>Basically, any associated form closes and the navigator uses the selected object as its value.</p> <ul style="list-style-type: none">• List View — When in list view mode, the objects are listed alphabetically from the top of the view pane to the bottom. If the list reaches the bottom of the navigator, it moves to the right and starts a new column of items. The object representations in this view are simply a graphical map representing the item's type, and the name of the item. It is impossible to get a vertical scroll bar while in List View mode. This is the default view for the navigator.• Details View — When in details view mode, the objects are initially listed alphabetically from the top of the view pane to the bottom. A vertical scroll bar is added if there is not enough room to display all of the items in the pane. There are multiple columns in the detail view, which contain additional information about each file object -- File Name, File Type, Date Modified, Size, Permissions, and Lock. The view can be sorted by any of these columns by clicking their respective header at the top of the view.

Table 2-3. Browse for Object Dialog Box Options

Field	Description
Buttons	<p>Following are the iconic button options:</p> <ul style="list-style-type: none"> • Back Directory button — This button changes the active navigator directory to the directory that was active previously. If there has been no change of directory, this button is disabled. When in 'Location Map' mode, this button changes to the directory that was active directly before 'Location Map' mode was activated. • Up Directory button — This button changes the active navigator directory to the physical parent of the current active directory. If the current directory is the root of the disk, this button is disabled. This button is also disabled in 'Location Map' mode. • New Directory button — This button prompts you for a directory name. If that name is entered and accepted, it creates a directory in the current active navigator directory with the specified name. This button is disabled when the 'Location Map' mode is active. • Browse Location Map button — This button enters 'Location Map' mode, which is essentially a browse operation on the active location map. It turns the entries in the active location map into 'directory' entries, which can be browsed into by double-clicking them. This button is disabled if no location map has been loaded into memory. • Change View button — This button changes the view pane display mode. There are currently two available modes: List and Details.
Search field	<p>The Search field allows you to filter the view pane for certain text combinations. The view pane checks the search field, and if it has any text, removes any objects from its view that do not have the entered text somewhere within the object name. The search field only uses basic text comparison, and it does not support wildcards or any form of regular expressions. For example, if a directory contains objects 'foo', 'bar', and 'baz', and the user types 'ba' into the search field, the 'foo' object is filtered out and the 'bar' and 'baz' objects remain. If the search field has text inside of it, a blue 'X' icon appears to the right of the field. If a user clicks on this icon, all text in the search field is removed.</p>

The list area, scroll bars, title bar, and navigation buttons of the object browser dialog work in the same manner as those on the Pyxis Project Manager navigator. However, the object browser

dialog does not have an iconic mode. It always displays design objects in a list with a small icon.

Pathname handling in the object browser dialog is the same as pathname handling in both the iconic and list navigators. Both hard and soft pathnames are accepted as input. The object browser dialog attempts to convert all hard pathnames to soft pathnames and displays the results. The object browser dialog returns all selected object pathnames as soft pathnames, if possible.

When you enter a relative pathname into the object browser dialog, the value returned by the `$get_navigator_directory()` function is used to convert the relative path to an absolute pathname.

As with any dialog box control, you can integrate the object browser dialog into one of your dialog boxes, if you want.

Related Topics

[Navigator Components](#)

[\\$get_navigator_directory\(\)](#)

[\\$form_navigator_gadget\(\)](#)

[Design Management with Location Maps](#)

Pathnames in the Pyxis Project Manager

A pathname is any path within some containment hierarchy, such as the file system's directory structure, that gives you the location of an object in that hierarchy.

An example path might be something like “/user/elec_engr/design1” which is the pathname of the object “design1”.

In the Pyxis Project Manager, the terms interoperable pathname, equivalent pathname, hard pathname, soft prefix, soft pathname, absolute pathname, and relative pathname have the following meanings:

- An *interoperable pathname* is a pathname that leads you to the same or equivalent object within a given context, regardless of which workstation you start from on your network. In addition, if it leads you to a given object at one time, it will lead you to the same object again any time later.
- A pathname is considered to be *equivalent* to another pathname when both pathnames locate the same object within a given context.
- A *hard pathname* is any pathname that begins with the slash (/) character. A hard pathname is not necessarily interoperable. The network and file system in which the pathname exists determines whether it is interoperable.
- A *soft prefix* is a string that precedes all other characters in a pathname, begins with a dollar sign (\$) character, continues with any alphanumeric character, and is either immediately followed by a slash (/) character or does not have any characters following

it at all. You can define a soft prefix's associated value with either a shell environment variable or with an entry in a location map.

- A *soft pathname* is any pathname that begins with a soft prefix. A soft pathname is not automatically interoperable. However, when you administer a location map or define environment variables for use as soft prefixes, you can guarantee interoperability.
- An *absolute pathname* is either a hard or a soft pathname, and specifies the pathname to an object.
- A *relative pathname* is any pathname whose initial character is not a slash character (/) or a dollar sign character (\$).

The Pyxis Project Manager uses location maps to display soft pathnames, if possible. Functions that return pathnames also return soft pathnames, if possible. You can convert a soft pathname to a hard pathname, by using the `$$get_hard_name()` function. You can convert a hard pathname to a soft pathname, by using the `$$get_soft_name()` function.

You can use hard, soft, absolute, or relative pathnames to manage and navigate to your designs in the Pyxis Project Manager. To maximize your performance, you should use soft pathnames rather than hard pathnames, whenever possible.

Regardless of which type of pathname you use, the transcript area always displays pathnames exactly as you entered them.

If you choose to work without a location map, you are responsible for ensuring that the pathnames in your network configuration can be resolved across workstation boundaries.

Note

 Placing a hard pathname in your location map does not automatically convert references, with those strings in their pathnames, to their corresponding soft pathname. The conversion of hard pathnames to soft pathnames occurs only when you create references or explicitly change references.

In the Pyxis Project Manager navigator window, you can use the `$get_navigator_directory()` function to return the value of the *current navigator directory*. The Pyxis Project Manager uses this value in the navigator window to convert relative pathnames to absolute pathnames.

In all other Pyxis Project Manager windows, you can use the `$$get_working_directory()` function to return the value of the *MGC working directory*. The Pyxis Project Manager uses this value to convert relative pathnames to absolute pathnames in all windows except the navigator window.

Table 2-4 summarizes the Pyxis Project Manager functions that resolve or report information about pathnames.

Table 2-4. Pyxis Project Manager Pathname Functions

Function	Description
\$\$fix_relative_path()	Generates an absolute pathname from a relative pathname.
\$\$get_hard_name()	Returns the hard pathname equivalent of any pathname.
\$get_navigator_directory()	Returns the soft pathname of the current navigator directory.
\$get_navigator_directory_hard()	Returns the hard pathname of the current navigator directory.
\$\$get_soft_name()	Returns the soft pathname equivalent of any pathname.
\$\$get_working_directory()	Returns the value of the MGC working directory.
\$\$is_relative_path()	Indicates if the specified pathname is a relative pathname.
\$\$resolve_path()	Resolves a specified pathname into a full hard pathname that does not have a symbolic link in its path.
\$\$set_working_directory()	Changes the value of the MGC working directory.
\$set_working_directory()	Changes the value of the MGC working directory interactively.

Related Topics

[\\$\\$get_hard_name\(\)](#)

[\\$get_navigator_directory\(\)](#)

[Working Without a Location Map](#)

[\\$\\$get_soft_name\(\)](#)

[\\$\\$get_working_directory\(\)](#)

Design Management with Location Maps

As a designer, you generally use a suite of applications in dealing with an entire set of design objects that all belong to one design.

To manage your design project, it is sometimes necessary to move, copy, release, archive, or otherwise manipulate all or part of your design. Since the design objects within your design relate to each other, they often hold references to each other. The pathnames on those references

capture the relationship among them. However, to manage your design, you may need to change the location of some or all of the design objects in it, thus breaking the relationships described by the references. Therefore, you need a way to find and correct all the references that are broken by moving design objects.

Additionally, in order to find a file in your computer network, the file system requires a starting point from which to begin its search. This location is known as the *network root*. Different workstations have different ways to indicate the network root in a pathname, so the pathname to the same object may vary on different workstations. You need a way to find any design object that should be visible to you from any workstation on your network. You also need to be able to find that same object using the same pathname at any later time. Pathnames that satisfy these requirements are *interoperable*.

The pathnames stored in your design data (often in references) are not all *interoperable*. If some of your pathnames are not interoperable, you could have trouble finding related objects in certain environments, or in guaranteeing their stability over time. Worse yet, you could find different objects, depending only on which workstation you are using.

Mentor Graphics supports interoperable pathnames by providing *location maps*. Location maps use soft prefixes to enable you to update references, by changing the location map or the relevant environment variables without touching the design objects themselves, and to navigate through your file system from a well-defined starting point.

Location maps manage pathnames in both heterogeneous and homogeneous environments. The following list enumerates some of the benefits of location maps:

- Allow you to retrieve design data without a rigidly structured hierarchy of links and network mounts
- Provide flexible placement of Mentor Graphics Tree components
- Allow pathnames that lead to data without regard for platform types
- Allow Mentor Graphics applications to adapt to your network
- Allow you to set up projects without root or network administrator privileges
- Provide easily administered redundancy for shared design components, such as parts libraries
- Allow data to be moved without changing internal design references

The Location Map	73
Working with Location Maps	77
Following References	83
The MGC Working Directory	85
Moving the MGC_HOME software tree	87
Working Without a Location Map	87
Location Map Administration	89

The Location Map

A *location map* defines the mapping between interoperable soft prefixes and their associated hard pathnames which identify where the design data is actually stored, from the perspective of one or more workstations.

Location Map File Format	73
Pathname Conversion Using Location Maps	74
Example of How a Location Map Resolves Pathnames	75

Location Map File Format

The location map is an ASCII file that associates hard pathnames with soft prefixes.

A location map consists of a header, followed by a series of entries that define the pathname mappings, with comments and delimiters interspersed as desired. An example location map is presented in Figure 2-13.

The location map file format guidelines are as follows:

- The first line of the location map file is always the header and consists of the string “MGC_LOCATION_MAP_2”. This string may be followed by the string “force”. The strings must be separated by one or more spaces.

The design data management system (DDMS) reads this line to determine the appropriate file format version of the location map. The optional “force” string tells DDMS that when an attempt to convert a hard pathname to a soft pathname is not successful, DDMS should issue an error, rather than store the hard pathname as a reference. Also, when you use force, you restrict yourself to entries in the location map.
- Each soft prefix is identified by an initial dollar sign (\$) in the first column. Because soft prefixes function similarly to shell environment variables, they must meet the standards for shell environment variables, which means that you should avoid shell special characters, such as a period (.) or a question mark (?) in your soft prefixes. The dollar sign must be followed first by any alpha character and then any combination of alphanumeric characters is allowed, including the underscored character (_); spaces are not allowed. Soft prefixes can contain a maximum of 1024 characters. By convention, all alpha characters should be capitalized. A soft prefix cannot be a null string.
- Each hard pathname begins with a forward slash (/) in the first column, is followed by any hard pathname that your network can resolve, and maps to the first soft prefix that precedes it in the map. Hard pathnames can contain a maximum of 1024 characters. Spaces are not allowed in hard pathnames.
- All comment lines begin with a pound sign (#) in the first column and can be followed by any characters. A number sign in any other column is not interpreted as a comment. Blank lines and comment lines can appear anywhere in the file after the header line, and they are always ignored.

- Any character that follows the first whitespace on any given line is ignored.
- A *location map entry* consists of a soft prefix and 0 to 20 hard pathnames, that are associated with the soft prefix.

Related Topics

[Pathname Conversion Using Location Maps](#)

[Example of How a Location Map Resolves Pathnames](#)

Pathname Conversion Using Location Maps

Soft prefixes, along with the hard pathnames for accessing those directories in your network, are entries in the location map.

Typically, design data main directories are the locations of libraries or of the main partitions of your design projects.

Each soft prefix may have 0 to 20 hard pathnames associated with it, although one hard pathname for each soft prefix is most common. Each soft prefix and its associated hard pathnames represent the same or equivalent location. Additionally, environment variables that share a common name with a soft prefix in the location map, affect pathname conversion. If a soft prefix has zero hard pathnames associated with it, its hard pathname must be defined by the value of an environment variable.

Normally, when soft-to-hard pathname conversion occurs, the soft prefix is mapped to the first hard pathname. However, when an environment variable of the same name as the soft prefix exists, the conversion of soft-to-hard pathnames is performed by comparing the value of the environment variable with the value of the first hard pathname associated with the soft prefix. If their values differ, the prefix is mapped to the value of the environment variable.

When hard-to-soft pathname conversion occurs, the prefix of the hard pathname is matched to a hard pathname in the location map, and the prefix in the hard pathname is replaced by the soft prefix associated with the matching hard pathname in the map. A match occurs only if the pathname you entered matches the hard pathname in the map completely up to the delimiting character (/). For example, the pathname */abc/def/ghi* would match the location map hard pathnames */abc*, */abc/def*, and */abc/def/ghi*, but it would not match the hard pathnames */ab* or */abc/de*.

Related Topics

[Reading the Location Map into Memory](#)

[Following References](#)

[Creating References](#)

Example of How a Location Map Resolves Pathnames

The hard pathnames are assumed to be valid pathnames in your network environment to the location represented by their associated soft prefix. Together, each soft prefix and its associated hard pathnames create a single entry in the location map.

The sample location map in Figure 2-13 contains soft prefixes for a parts directory, a project directory, and two libraries.

The first line of the sample map file contains the string “MGC_LOCATION_MAP_2”, followed by the string “force”. This header line identifies the version of the map file format. The “force” string specifies that attempts to convert a pathname from a hard name to a soft name must be successful or an error should be issued. Without the “force” string, if a pathname cannot be converted from hard to soft, then the hard pathname are used. If you want to prevent hard pathnames from ever being stored as design object references, you should use the “force” string in the location map header line.

Each successive line of a map file is always a blank line, a soft prefix, a hard pathname, or a comment line.

Figure 2-13. Sample of a Location Map File

```

MGC_LOCATION_MAP_2 force      --> Location Map Header
                           --> White space delimiter
#Project XXX Parts          --> Comment line
$PROJ_PARTS                 --> Soft prefix
/project/XXX/parts          --> Hard pathname for $PROJ_PARTS
/user/rjones/projXXX/parts   --> Hard pathname for $PROJ_PARTS
                           --> White space delimiter
# Project XXX Other Repository --> Comment line
$PROJ_XYM                   --> Soft prefix
/home/project/XXX            --> Hard pathname for $PROJ_XYM
                           --> White space delimiter
#MGC Generic Library         --> Comment line
$MGC_GENLIB                 --> Soft prefix
/srvr4/libraries/mgc_genlib --> Hard pathname for $GENLIB

#NCR Library                 --> Comment line
$NCR                         --> Soft prefix

```

Comment lines, which provide documentation about an entry, usually precede every soft prefix entry in the map. In the sample location map, the soft prefix \$PROJ_PARTS is preceded by the comment line “#Project XXX Parts”.

The first soft prefix in this map is \$PROJ_PARTS, which has two hard pathnames associated with it: /project/XXX/parts and /user/rjones/projXXX/parts. On the workstation this location map resides on, both of these pathnames are valid paths to the location represented by the \$PROJ_PARTS soft prefix. Together, they comprise the first entry in the in the location map.

If you add a reference to /project/XXX/parts/d1, using this location map, the pathname is mapped to a soft pathname before it is stored in the reference. During the hard-to-soft pathname

conversion, the location map is searched for a hard name that matches `/project/XXX/parts/d1`. The first match `/project/XXX/parts` is located under the `$PROJ_PARTS` prefix. As a result, the `$PROJ_PARTS` soft prefix is substituted in place of `/project/XXX/parts` and the resulting pathname `$PROJ_PARTS/d1` is stored in your reference. In this way, the location map allows you to store a reference whose pathname is interoperable.

When you attempt to use the reference to find the object, the location map is searched for the soft prefix `$PROJ_PARTS`. The *first* hard pathname associated with the `$PROJ_PARTS` soft prefix is `/project/XXX/parts`. The `/project/XXX/parts` pathname is substituted for `$PROJ_PARTS` in the reference. As a result, the reference refers to the object at location `/project/XXX/parts/d1`.

If all your references to objects in your parts directory are stored as soft pathnames and your parts directory is later moved to `/user/bradc/projXXX/parts`, your references are still valid. Instead of modifying each individual design object whose references target the parts directory, you simply edit the location map and change the hard pathnames for the soft prefix `$PROJ_PARTS` to the new hard pathnames which access the new location `/user/bradc/projXXX/parts`.

**Note**

The location map does not specify a search order as does the UNIX environmental variable PATH. Rather the location map serves as a lookup for soft-prefix entries.

In this location map example, the `$PROJ_PARTS` soft prefix has two hard pathnames, which are alternate paths to the location represented by `$PROJ_PARTS`, associated with it. Multiple hard pathnames for a single soft prefix provide alternate routes to the same location. For example, the `/project/XXX/parts` and `/user/rjones/projXXX/parts` hard pathnames are paths to the same set of parts files.

Multiple hard pathnames for a single soft prefix also allow you to treat identical copies of the same object as though they actually are the same object. For example, you might treat copies of parts on different workstations as the same object. However, this practice is dangerous if the copies are not identical.

The `$PROJ_XYM` and `$MGC_GENLIB` soft prefixes each have one hard pathname. The `$NCR` soft prefix does not have any hard pathnames associated with it. You place soft prefixes without any hard pathnames associated with them, like the `$NCR` soft prefix, in the location map specifically to mandate that an environment variable of the same name must exist.

You might add the `$NCR` soft prefix to your location map without entering any associated hard pathnames, if you want to remove the need to modify your location map every time the `$NCR` library is moved. Whenever the system administrator moves the location of the `$NCR` library, he resets the value of the `$NCR` environment variable to the current pathname of the `$NCR` library, in a script that all users execute when they log in. This eliminates the need for individual users to modify the `$NCR` entry in their location map. When you attempt to access the reference

pathname containing the `$NCR` soft prefix, the environment variable `$NCR` returns the correct pathname value, in the absence of any associated hard pathnames in the map.

Related Topics

[Following References](#)

Working with Location Maps

Location maps provide you with a way to manage soft prefixes and interoperability, and serve you by automating some pathname processing.

When you are using a location map, you should enter soft pathnames directly into the application whenever you can. Soft pathnames are usually shorter than their equivalent hard pathnames, and entering them directly minimizes pathname processing.

The design data management system (DDMS) relies on the customized location maps that you create. When you work in an environment that uses location maps, you need to understand the following items:

- How the DDMS determines which location map file to use
- How the DDMS processes location map files
- How and when location maps are read into memory
- How the DDMS uses location maps to convert pathnames

Location Map Location	77
Including Multiple Location Maps	78
Parts Library Filtering	79
Reading the Location Map into Memory	80
Updating the Location Map	82
Location Maps and Projects	83

Location Map Location

The DDMS finds the location map that is to be used in your Pyxis Project Manager session by reading the value of the `MGC_LOCATION_MAP` environment variable. You can set the value of the `MGC_LOCATION_MAP` environment variable in your login file.

Whenever a location map is needed, the DDMS system searches for it in the following order:

1. `./mgc_location_map`
2. `$HOME/mgc/mgc_location_map`
3. `$MGC_HOME/etc/mgc_location_map`

4. \$MGC_HOME/shared/etc/mgc_location_map

This search hierarchy is invalid if the MGC_LOCATION_MAP environment variable is set. The environment variable overrides the precedence of the search hierarchy. To use the search hierarchy *do not set* MGC_LOCATION_MAP.

You can read a location map explicitly by executing the \$\$read_map() AMPLE function, which allows you to specify the map location directly.

You can also use the \$\$get_location_map() AMPLE function to retrieve the current in-memory location map. This function provides you with the pathname of the location map used to generate the in-memory map; it also reports each entry in the map and whether the hard pathnames associated with a soft prefix result from an environment variable value or from a location map entry.

Related Topics

[\\$read_map\(\)](#)

[\\$\\$get_location_map\(\)](#)

Including Multiple Location Maps

You can include multiple location maps (one inside another) by using the INCLUDE statement followed by the pathname to the included location map.

By doing this you can simplify the maintenance of location maps. You can keep separate location maps for parts libraries or specific projects. Whenever the INCLUDE statement is called, the application reads in that location map. Soft prefixes and their associated hard pathnames from the included map are valid for the design as if they were listed in the current location map.

For example, if you are looking for \$PARTS/nor2, then you do not need to have an entry in your personal location map for \$PARTS/nor2 but you can have a pathname associated with another map that does have the entry for \$PARTS/nor2.

Remember, the location map reads entries in order from top to bottom. If there is a duplicate soft prefix name the first prefix is used and any duplicate prefix is ignored, this also happens if the duplicate softpath is in another location map found by the INCLUDE statement. Therefore, the placement of the INCLUDE statement should be deliberate in order to maximize its usage. For example:

```
MGC_LOCATION_MAP_2
$C321_LAYOUT
/usr/project/c321/layout
$NCRLIB -t LIBRARY
/usr/library/in_house
$SCHEMATIC_LIB -t LIBRARY
/usr/library/schematic
# Include the shared location map for parts libraries
INCLUDE /usr/project/libraries/mgc_location_map
```

Related Topics

[Private Location Maps](#)

Parts Library Filtering

Parts library filtering can be achieved by using the key operator `-t(ype) LIBRARY` after a designated soft prefix name.

These key operators can be used for selecting out parts in a configuration, based on the library with which they are associated, using the configuration popup **Select > By Library**. The following shows a location map with this feature:

```
MGC_LOCATION_MAP_2
$C321_LAYOUT
/usr/project/c321/layout
$NCRLIB -t LIBRARY
/usr/library/in_house
$SCHEMATIC_LIB -t LIBRARY
/usr/library/schematic
#Include the shared location map for parts libraries
INCLUDE /usr/project/libraries/mgc_location_map
```



Note

The `-t LIBRARY` switch requires a hardpath name. Soft paths without associated hard paths are not supported with this switch. Environment variables do not overwrite soft pathnames that include the key operator `-t LIBRARY`.

Related Topics

[Working with Location Maps](#)

Reading the Location Map into Memory

The design data management system (DDMS) must read a location map file into memory, before it can use the information contained in it.

When the DDMS reads the map file into memory, it also looks for relevant environment variables and adjusts the entries according to their value. The location map information that resides in memory is used for pathname conversion.

The DDMS reads the location map into memory automatically when you invoke the Pyxis Project Manager, or any other application. If you edit your location map file while your application is running, those changes are not automatically available. You must read your location map into memory again to be able to access the modified location map.

When the DDMS determines the correct location map to use, it reads it into memory by processing each entry in the location map file. The DDMS processes entries by searching for shell environment variables that match the names of the soft prefixes in the map. For each entry, one of four actions occurs, depending upon the existence of a matching shell environment variable:

1. If no environment variable matches a given soft prefix from the map, the entry from the map file remains unchanged.
2. If an environment variable matches a given soft prefix, the value of the environment variable is compared against the first hard pathname associated with the soft prefix in the location map file.
 - If the hard pathnames match, the entry from the location map file remains unchanged.
 - If the values differ, the entry is replaced. The value of the environment variable becomes the single hard pathname in the entry. In this way, the environment variable overrides the location map file.
3. If a soft prefix in the location map has zero hard pathname entries associated with it and there is a matching environment variable, the entry is built in memory, using the value of the environment variable only.
4. If a soft prefix in the location map has zero hard pathname entries associated with it and there is *not* a matching environment variable, an error message is displayed. You must correct the problem before proceeding. You cannot use the location map you now have in memory.

If a soft prefix's associated hard pathnames are overridden by the value of an existing environment variable when the map is read into memory, the DDMS places the comment "(overridden)" next to the entry in the location map. This comment signifies that the hard pathnames associated with a soft prefix in the location map were overridden by the value of an environment variable with the same name as the soft prefix. However, this comment is also

placed in the location map when you override an entry by using the `$change_location_map_entry()` function, as discussed in the following text.

Table 2-5 illustrates how the entry for the soft prefix `$PROJ_PARTS` is created in memory in varying cases.

Table 2-5. Example of Processing a Soft Prefix

Matching Shell Environment Variable	Location Map File Entry	Location Map Entry in Memory
None	<code>\$PROJ_PARTS</code> <code>/home/project/XXX/parts</code> <code>/user/rjones/projXXX/parts</code>	<code>\$PROJ_PARTS</code> <code>/home/project/XXX/parts</code> <code>/user/rjones/projXXX/parts</code>
<code>\$PROJ_PARTS=</code> <code>/home/project/XXX/parts</code>	<code>\$PROJ_PARTS</code> <code>/home/project/XXX/parts</code> <code>/user/rjones/projXXX/parts</code>	<code>\$PROJ_PARTS</code> <code>/home/project/XXX/parts</code> <code>/user/rjones/projXXX/parts</code>
<code>\$PROJ_PARTS=</code> <code>/project/XXX/parts</code>	<code>\$PROJ_PARTS</code> <code>/home/project/XXX/parts</code> <code>/user/rjones/projXXX/parts</code>	<code>\$PROJ_PARTS</code> <code>/project/XXX/parts</code>
None	<code>\$PROJ_PARTS</code>	Error
<code>\$PROJ_PARTS=</code> <code>/project/XXX/parts</code>	<code>\$PROJ_PARTS</code>	<code>\$PROJ_PARTS</code> <code>/project/XXX/parts</code>

You should usually run your applications in shells where you have not defined any environment variables that match the soft prefixes in your location map, except for environment variables that match soft prefixes which have zero hard pathnames. Because environment variable values override the location map file entries, these variables can alter the way the DDMS interprets the location map file, initially. If the hard pathnames for a soft prefix are different from the value of the environment variable with the same name, as is the case when a soft prefix has multiple hard pathnames associated with it, the application uses only the environment variable value.

Additionally, when you change your location map, you are unable to see those changes until you exit your application, change or eliminate the environment variable, and run the application again.

Due to errors in the location map file, DDMS may not be able to successfully read the location map into memory. When DDMS encounters an error, it displays an error message in the transcript and displays a dialog box with the following options:

- **Edit the location map.** The location map file is opened in a Notepad window for editing. You should correct the errors in the map file, save the file, and use `$read_map()` to read the map file again.

- **Exit the application.** The `$close_session()` function is called to close the application session. You should correct the errors in the map file and re-invoke the application.
- **Continue.** The application session continues. If a valid location map was in memory prior to encountering the error, then that location map is still active; otherwise, the application session continues with no location map active.

The error message in the transcript identifies the specific line in the location map that caused the error to help you to edit the map file.

Related Topics

[Updating the Location Map](#)

Updating the Location Map

You should *seldom*, if ever, need to change an entry in the location map while it is in memory.

Normally, you should make your changes to the ASCII location map file and re-read the whole map into memory using the `$$read_map()` function. You may use the `$read_map()` function if you are in the Pyxis Project Manager and want to work on another project, without exiting the session.

Caution



You should never re-read a location map when you are in an application and working on design data. If you do so, you risk violating the integrity of your design. Although location maps can be changed and re-read without exiting an application, you should only do so when you are beginning work on another design or project.

However, the `$$set_location_map_entry()` function does allow you to change an entry in the in-memory location map for your Pyxis Project Manager session. If you choose to execute this function, you supply a soft prefix that corresponds to the entry that you want to change in the map and a new hard pathname that you want to associate with that entry. If you fail to supply the hard pathname as an argument to this function, or if the soft prefix that you supply does not already exist in the in-memory map, an error is returned and you cannot add these new entries to your location map. If an entry for the soft prefix is located in the map, the first hard pathname for that soft prefix and the hard pathname you supplied are compared. If the two pathnames are different, the new hard pathname you supplied replaces all hard pathnames associated with that soft prefix and the comment “(overridden)” is placed next to the entry in the map, signifying that the entry is modified since the map was read into memory. If the two pathnames are the same, no action takes place.

Related Topics

[\\$read_map\(\)](#)

[\\$\\$set_location_map_entry\(\)](#)

Location Maps and Projects

While creating and maintaining projects using the Project Navigator feature of Pyxis Project Manager, location maps are automatically managed by Pyxis Project Manager.

Related Topics

[Working with Location Maps](#)

Following References

Because the file system does not recognize soft pathnames, a reference path must be converted to a hard pathname before you can use it to find an object. If the reference pathname is already hard, no conversion takes place.

When you follow a reference in the Pyxis Project Manager, if its pathname begins with a “\$”, the first element of the pathname string is assumed to be either a soft prefix in the location map or an environment variable. In this case, the DDMS searches for a soft prefix that matches the first element of the pathname string; the prefix may be in the location map or it may be an environment variable.

If a match is found, the first hard pathname associated with the soft prefix is substituted in place of the element in the pathname string and the resulting hard pathname is traversed. If a match is not found, a search for a matching environment variable is performed. If an environment variable matching that element is found, its value is substituted in place of the element in the pathname string and the resulting hard pathname is traversed. If neither a soft prefix nor an environment variable match the first element of the pathname string, an error occurs.

Creating References	83
Conversion of Hard References to Soft References	85

Creating References

When you create a reference or a design object, or when you want to cache a pathname in your application, the pathname you enter is converted to a hard pathname, if necessary.

The pathname conversion is a two-step process.

If the pathname string is a soft pathname, the DDMS searches the location map for a matching soft prefix. If there is a match, the pathname is accepted and stored in the reference.

 Note	Placing a hard pathname in your location map does not automatically convert references with those strings in their pathnames to their corresponding soft pathname. The conversion of hard pathnames to soft pathnames occurs only when you create references or explicitly change references.
---	---

To illustrate the successful creation of a reference using a soft pathname, assume that your location map includes the following entry:

```
$XYZ DESIGN
/usr2/project_xyz/my_design
```

If you add a reference to *\$XYZ DESIGN/cmp/mmu*, the “\$” identifies the first element of the pathname as a soft prefix. DDMS searches the location map for an occurrence of *\$XYZ DESIGN*. Since the soft prefix *\$XYZ DESIGN* exists in our location map example, the search is successful and the pathname is accepted and stored in the reference.

If the pathname string is a hard pathname, the DDMS searches the location map for a hard pathname that matches the leading pattern of your hard pathname. If there is a match, the DDMS substitutes the soft prefix from the location map into the matching portion of your pathname string. If there is not a match, the DDMS determines what to do based on the presence of the “force” string in the location map header line. If the “force” string is present, the DDMS issues an error; if the “force” string is absent, the pathname string is accepted as you entered it, or in its backwardly-compatible equivalent form.

To illustrate the successful creation of a reference using a hard pathname, assume that your location map includes the same entry, as shown in the previous example:

```
$XYZ DESIGN
/usr2/project_xyz/my_design
```

If you add the reference */usr2/project_xyz/my_design/cmp/ram* to a design object, the pathname is converted to a soft pathname in the following way. The DDMS compares the reference pathname to the hard pathnames in the location map. The reference */usr2/project_xyz/my_design/cmp/ram* matches the hard pathname */usr2/project_xyz/my_design* in the location map. The soft prefix *\$XYZ DESIGN* is substituted in place of the */usr2/project_xyz/my_design* string and the resulting pathname *\$XYZ DESIGN/cmp/ram* is stored in the reference.

If the pathname string does not begin with either a dollar sign (\$) or a slash (/), the DDMS interprets the pathname string as a relative pathname and processes it in the following way:

1. DDMS appends the MGC working directory to the beginning of the relative pathname.

Appending the MGC working directory to the pathname creates an absolute pathname. If the absolute pathname is a soft pathname, DDMS converts it to a hard pathname before performing step 2.

2. DDMS processes the absolute pathname to remove any “./” and “/..” sequences.

The DDMS processes the absolute pathname by appending the MGC working directory to the beginning of relative elements in the path and flattening *../* elements without resolving any links in the pathname.

3. DDMS converts the pathname string to a soft pathname, if necessary, based on the location map in memory.

To illustrate creating a reference using a relative pathname, assume that you again have the same location map entry as shown previous, and you enter the following commands from a C shell:

```
% setenv MGC_WD /usr2/project_xyz/my_design/cmp  
% cd $MGC_WD  
% $MGC_HOME/bin/da
```

You create a reference using the relative pathname *ram*. Your MGC working directory */usr2/project_xyz/my_design/cmp*, returned by the MGC_WD environment variable, is appended to the beginning of the relative pathname resulting in the path */usr2/project_xyz/my_design/cmp/ram*. Because this is a hard pathname, it is converted to the soft path *\$XYZ_DESIGN/cmp/ram*, via the location map, and stored in the reference.

Related Topics

[The MGC Working Directory](#)

Conversion of Hard References to Soft References

If your design object or configuration object was created before the introduction of location maps, the references on the object are hard references.

The functions `$convert_object_references()` and `$convert_configuration_references()` use the entries in the current location map to automatically change the references on an object from hard to soft pathnames.

Since these functions use the current location map to perform reference pathname conversions, they cannot execute unless a location map is loaded. If you attempt to execute either of these functions without a location map loaded, a message is displayed stating that you must first use the `$read_map()` function to load a location map before performing the conversions.

You can specify whether warnings are reported for any references that were not successfully converted to soft pathnames. By default, this information is not reported.

Related Topics

[\\$convert_object_references\(\)](#)

[\\$convert_configuration_references\(\)](#)

The MGC Working Directory

To resolve the location of an object, DDMS must convert relative pathnames to absolute pathnames.

This type of conversion occurs automatically whenever you create a reference or a design object, or when you want to cache a pathname in your application.

The DDMS sets the initial value of the *MGC working directory*, when you invoke the Pyxis Project Manager. To initialize the MGC working directory, the DDMS uses the value of the MGC_WD environment variable. You should always set your MGC_WD environment variable in your login script. If your MGC_WD environment variable is not set, the DDMS uses the value of the UNIX current working directory, which is unreliable and cannot be guaranteed to be interoperable.

If you work on multiple projects simultaneously, you may want to create a shell startup script for each of those projects and include in it the command “cd \$MGC_WD” to synchronize the DDMS and UNIX pathname resolution mechanisms.

You can use the \$\$set_working_directory() AMPLE function whenever you want to change your MGC working directory, and the \$\$get_working_directory() AMPLE function whenever you need to know what it is.

Caution

 You should never specify a pathname that starts with “\$MGC_WD”. This is an improper use of the MGC_WD environment variable. Attempting to use \$MGC_WD as a soft prefix causes an error because \$MGC_WD is not typically a soft prefix in the location map. Although you can, theoretically, add the soft prefix \$MGC_WD to the location map, this is impractical since most users do not have the same working directory. Furthermore, it is unnecessary because you do not need to specify that a relative pathname is resolved using the value of the MGC working directory. Relative pathnames are automatically resolved using the value of \$MGC_WD.

You should never set your MGC_WD environment variable to \$HOME or \$PWD. If you do so, you are solely responsible for the interoperability of your data.

Setting the MGC_WD environment variable to the value of either the HOME or PWD environment variable has a major impact on the server environment and on data interoperability. For example, setting MGC_WD to \$HOME causes any subsequent designs that you create to have the \$HOME environment variable at the beginning of the reference, as shown below:

\$HOME/design/schematic/sheet1

If all the user's have their MGC_WD environment variable set in the same way, only the person who has this design in their home directory can access it. If any other user tries to access or change the design, references are broken and data is not accessible.

Additionally, this implementation defeats the purpose of concurrent engineering, since a copy of the design would have to reside in every \$HOME or \$PWD directory.

Mentor Graphics does not provide support for setting your MGC_WD environment variable to \$HOME or \$PWD. Furthermore, Mentor Graphics does not take responsibility for data interoperability in this implementation.

Related Topics

[\\$\\$get_working_directory\(\)](#)

[\\$set_working_directory\(\)](#)

[Private Location Maps](#)

Moving the MGC_HOME software tree

Moving an installed tree using the operating system invalidates location map paths used for the MGC standard libraries.

 **Caution**
It is strongly recommended that you DO NOT move or rename an installed software tree from its originally installed location. The Mentor Install Program (MIP) cannot validate, update, or patch the tree if it has been moved.

Procedure

If a tree has been moved in this manner, do one of the following options.

1. Uninstall the MGC_HOME tree first and then reinstall it in the new location. This is the recommended solution.
2. Update the location map paths within the MGC location map template located at \$MGC_HOME/shared/etc/appl/base/mgc_map_template. This location map is generated as part of the installation process and is used by the Pyxis Custom Design flow to provide access to these libraries. The MGC standard libraries are located within \$MGC_HOME/mgc_icstdlib.

Related Topics

[Pyxis Administrator's Guide](#)

Working Without a Location Map

To operate without a location map, set the value of your MGC_LOCATION_MAP environment variable to the string “NO_MAP”. If you choose to operate without a location map, you assume responsibility for the interoperability of pathnames in your network.

If you choose to use environment variables at the start of your pathnames, you must set the environment variables which are used as soft prefixes in your design, and you must maintain

these environment variables so that they are consistent and available to all other users in your project team.

Additionally, if you work without a location map, you *must* set shell environment variables to access your V8.X libraries.

Note

 Because the internal references of V8.X library parts contain soft prefixes, you must set shell environment variables for V8.X libraries in order to access them. Mentor Graphics strongly recommends that you use location maps to facilitate your access to V8.X library parts.

If you use primarily hard pathnames, you are responsible for ensuring that the hard pathnames you put in your design data are interoperable.

Although working without a location map has the least overhead, it also has the least control or aids. Working without a location map is not generally recommended. However, users who work alone or in very small groups, or users whose network configuration is homogeneous may find this to be an acceptable option.

Pathname Conversion	88
Relative Pathnames	88

Pathname Conversion

When you work without a location map, pathnames are only processed when you attempt to access a file.

If the pathname begins with a (\$), the initial element of the name is assumed to be an environment variable and the variable's value is substituted in its place. All other pathnames are accepted exactly as entered.

Hard pathnames and pathnames that begin with an environment variable are stored in references exactly as you typed them. Nothing is done to change any full reference pathnames that you enter, prior to storing them.

Related Topics

[Relative Pathnames](#)

Relative Pathnames

The first step in relative pathname conversion is the same if you are operating with or without a location map.

If you work on a design object in a given directory, and refer to the object with a relative name, the full pathname is generated by first appending the MGC working directory string to the front of the relative name, and then flattening out any “`../`” and “`.`” sequences.

Note

 If a pathname contains a link element that precedes either a “`.`” or “`..`” sequence, DDMS interprets the pathname slightly differently from UNIX. DDMS simply manipulates the original pathname string, while UNIX searches the file system to resolve the pathname. For example, given a pathname of the type `/original_prefix/link/..`, the DDMS returns `/original_prefix/link/` and UNIX returns `/resolved_prefix/leaf/`.

When you work without a location map, pathnames are not converted to soft pathnames prior to being stored. Therefore, if the full name that results from relative-to-absolute pathname conversion is not interoperable, you may have to use one of the Change References commands to manually modify the real pathname.

Related Topics

[The MGC Working Directory](#)

[\\$\\$get_working_directory\(\)](#)

[\\$\\$set_working_directory\(\)](#)

Location Map Administration

Location maps are very flexible and relatively easy to construct and use, but they require careful system management.

There are several types of location maps: master location maps, project location maps, and private location maps.

Master Location Map Administration	89
Project Location Map Administration	92
Private Location Maps	95
Server Location Maps	98
Location Map Naming Conventions	100

Master Location Map Administration

Each MGC customer should administer at least one master location map.

This map represents a central point for supporting interoperability at a work site, or across a reasonably localized computer network. The following text addresses the administrator of the master location map.

If you work for a large company that has multiple work sites or networks, your company may decide to distribute the administration of master location maps, by partitioning your computing

environments by network, work site, or some other reasonable means. As one of the master location map administrators, you must share the responsibilities outlined in this manual with the other partition administrators, and together you must coordinate activities among yourselves to provide consistency in the location maps throughout your company.

A master location map assists you in achieving the following two goals:

- Helps you coordinate the creation and use of soft prefixes at your work site, to prevent naming conflicts.

Your project location map administrators register the soft prefixes they want to use in their projects, with the master location map administrator. As the administrator, you must ensure that those names do not conflict with soft prefixes used in the data you purchase, such as MGC libraries, or with your company's own projects. You should also consider that your own customers may want to create data with their own soft prefixes, and leave them room in the name space.

- Helps you maintain a master record of what physical location each soft prefix represents in your file system.

Designers do not generally use master location maps. Instead, each project creates a project map by cloning the master map and cutting it down to the subset required for their project's work. The project administrator makes any necessary modifications to the hard pathnames in the project map, to ensure that the project team members can efficiently access the data they need.

The master location map guides the projects to the hard locations where the targets of the soft prefixes reside on your network. It is, therefore, the master record of where to find shared data at your work site.

Normally, the master location map simply contains one hard pathname for each soft prefix, which shows how to reach each target from a given example workstation. If such hard pathnames are not available to you, you can simply use some readable pseudo-pathname that designates the location. In that case, of course, no machine process can use the master map directly.

An administrator should be appointed to administer the master location map for the site or network. As the administrator you should coordinate the maintenance of the master location map, based on the following guidelines:

- You should set up and maintain your lines of communication with the project location map administrators, or the project team leaders, to support the following needs:
 - Project location map administrators must register their projects' soft prefixes for entry into the master location map, with you. You must manage the soft prefix name space to avoid name collisions.

- You must disseminate information about location map issues and changes to the project location map administrators, so that project location maps remain consistent with the master location map.

For example, notify them when a library is installed or moved, or when a network administration change occurs. Communicate both the specific nature of the change, and its impact upon location map usage.

- If your company decides to automate location map changes for the project teams at your site, you should be responsible for making the changes. In this case, you need to maintain a list of the project location map locations and the project location map administrators need to notify you of changes, in order to keep this list current.
- You should establish a standard fixed location for each master location map.

Your co-workers must be able to read the master location map easily, and to copy it to update their location maps. However, only you should have rights to change the master map.

- You should maintain a complete set of soft prefixes in the master location map.

The master location map should contain all the soft prefixes used at your work site. This set includes all the prefixes from the projects at your site, and all those required by the libraries you have installed, including MGC libraries.

The master location map is primarily an administration resource. You need to search it to perform some tasks related to it. Project teams also need to identify which soft prefixes are relevant to their needs. Therefore, you should assign a natural ordering to the soft prefixes in your master location map, such as alphabetical order. The primary purpose of the master location map is for human reference, so you should make it as easy as possible for people to find an entry.

- You should develop soft prefix naming conventions for your company.
- You should approve and register soft prefixes for entry into the master location map.

Soft prefix names probably originate with your project groups. However, the project administrators must submit those soft prefixes to you and you must approve and register any proposed soft prefix names before they become available globally in the master location map.

Before you approve a soft prefix name, you must ensure that it meets the following two requirements:

- The soft prefix must adhere to all your naming conventions for soft prefixes.
- The soft prefix cannot already be present in the master location map.

Enforcing these criteria may help you avoid naming conflicts between the soft prefixes used with your company and your customer's companies.

- You must assign hard pathnames for each soft prefix in your master location map.

- You should create and manage a location map for your shared server processes to use.

Related Topics

[Location Map Naming Conventions](#)

[Server Location Maps](#)

Project Location Map Administration

As a project location map administrator you should adhere to the same set of guidelines for specifying soft prefixes and hard pathnames, as the master location map administrator.

If you are a project location map administrator, you should be familiar with the guidelines listed in “[Master Location Map Administration](#)” on page 89 later in this chapter, before reading the following text.

If the workstations used by your project team are networked so that you cannot access all shared project data through a common set of hard pathnames, your project may need more than one project location map. If this is the case, file system pathnames that work on one set of machines do not work on another and you need one location map for each unique set of pathname conventions.

As the project location map administrator, you may choose to manage all of the location maps in your project yourself, or you may choose to require that each designer creates and maintains a private location map. In the later case, each designer is responsible for knowing the correct hard pathnames for alternate machines.

Just as with a master location map, the number and order of the entries in a project location map can affect data access performance. Although the reduced size of a project location map, in comparison to a master location map, yields performance benefits, you should not eliminate entries from your location map that your project needs, in order to further maximize performance. Doing so means that your project is unable to access essential data using soft pathnames.

As the project location map administrator, you should coordinate the maintenance of your project location map based on the following guidelines:

- You should set up and maintain your lines of communication with the master location map administrator.

This is the person from whom you must obtain a project prefix and with whom you must register your project’s soft prefixes. You must coordinate any changes to your project map that are visible outside your project. If you do not register your soft prefixes with the master map administrator, you probably cannot print or plot your project data. The master location map administrator notifies you of all changes to the master location map that might potentially affect your project. For example, the master location map administrator notifies you of changes to the master location map when a library or a portion of a project is moved or copied, or when a project adds a new soft prefix.

- You should maintain a current list of your project's team members and use it to perform the following tasks.
 - Identify yourself as the focal point for any project-wide issues which impact data access.
 - Disseminate any information about these issues that originates outside your project, and coordinate any related activities internal to your project. Notify your project team of changes to the master or project location maps.
 - If your project decides to automate project location map changes, you are responsible for automating the process. In this case, you need a list of your project team's private location map locations, and you should notify members of the team that you need to keep this list current. Alternatively, you may want to create and maintain all of your project's private location maps yourself, and require all project members to use one of them.
- You should guarantee the accessibility of your project's data.

The pathnames that you use in Mentor Graphics applications must direct any process that needs to see your data, including servers, to its location on your network. You are the person who is responsible for guaranteeing the accessibility of your project's data. If you use soft pathnames at your site, we recommend that you use location maps to help you ensure that this requirement is met.

- You should establish a standard fixed location for each of your project location maps.

Your project's team members must be able to read the project location maps easily, and to copy them to update their private location maps. Only you, and perhaps your master location map administrator, should have rights to change the project location map. You may need to tell the master map administrator the location of your project location map.
- You should obtain a common project prefix from your master location map administrator.

All the soft prefixes your project uses should start with the same sequence of characters. This sequence is a prefix appended to the beginning of your project's soft prefixes, and it uniquely identifies all the soft prefixes that originate from your project. Obtain this prefix from your master location map administrator.

You are largely free to manage the soft prefix name space that follows your common project prefix. However, you should check with your master location map administrator to see if any further company naming standards apply.

- You should ensure that your project location map contains the minimum set of soft prefixes.

At a minimum, your project location map should contain all the soft prefixes your project defines for the master map and all the soft prefixes for the libraries your project uses.

For accessing data outside your project, coordinate with your master location map administrator. If duplicates of libraries exist in your network, you need to find out which hard pathname to use in your location map. Making the correct selection helps you improve performance and optimize network data flow.

- You must register your project's soft prefixes.

To create a soft prefix for your project location map, the master location map administrator must approve the soft prefix name and register it. Only when the soft prefix is approved and registered does it become globally available. The principal goals of soft prefix registration are to ensure that there are not any naming conflicts in the master location map, and to enable all servers to access the related data.

You must also inform the master location map administrator of all the hard pathname mappings for each of your soft prefixes, for each of your machine types.

- You should ensure that each of your project's soft prefixes relates to a collection of data.

You should try to create soft prefixes for your project that correspond to logical groupings of data, such as a library that you might want to move or copy as a unit. All of the data that is tied to one soft prefix should be small enough to fit on one disk. At the same time, you do not want your project location map to be too fine-grained, as it would be if you assigned a soft prefix to each individual design object. You also do not want your map to be too coarse-grained. One soft prefix is probably appropriate for many small projects, but larger projects need more than one. For most projects, the number of soft prefixes should be less than the number of designers assigned to the project.

- You should only use one soft prefix per logical collection of data.

You should avoid creating multiple soft prefixes that are hierarchically related. You should not create soft prefixes that refer to subdirectories of the locations of other soft prefixes. If you do, you may need to change references when you move or copy data under that soft prefix.

- You should avoid creating soft prefixes that are hierarchically related.

You should not create soft prefixes that refer to sub-directories of the locations of other soft prefixes. If you do, you may need to change references when you move or copy data under that prefix.

- You should treat with care any hard pathnames that begin with “//”.

Although it is not actually incorrect to place a pathname beginning with “//” in your location map, it slows down pathname processing. If you want to show an equivalence, we recommend that you place the equivalence in a comment line of the location map, as shown in the following example:

```
# Project XXX Parts
$COMPANYID_PROJXXX_PARTS
/project/XXX/part
/user/rjones/projXXX/part
# //rjones/local_user/rjones/projXXX/part
```

Mentor Graphics provides you with considerable flexibility in setting up and managing your location maps. We do not absolutely require you to follow our guidelines and recommendations, but we urge you to do so. Although choosing not to follow our recommendations should not result in a loss of data, it may cause you some inconvenience.

Related Topics

[Master Location Map Administration](#)

Private Location Maps

A private location map is generally a project map that has been customized within strict limits. Each individual may manage a private location map. However, you usually avoid doing so unless your situation requires it.

Using a project location map eliminates your maintenance responsibilities. If you must use a private location map, be sure that you know why it is needed. If these circumstances change at a later time, you may be able to eliminate your private location map.

The following text addresses the individual who needs to administer a private location map. As the administrator of your private location map, you have the same responsibilities as a project location map administrator, with two exceptions. You are not responsible for communicating with the master location map administrator or for maintaining project-wide conventions.

Your private map is simply a customized project map that contains the hard pathnames you need during design development. As a private location map administrator, you should also adhere to the same set of guidelines for specifying soft prefixes and hard pathnames, as the master and project location map administrators.

Private Location Map Administration Guidelines

- You should set up and maintain your line of communication with your project location map administrator.

This person notifies you of changes to the master and project location maps that might potentially affect your private location map. If you need to change the project map, by adding a soft prefix for example, you must contact this person to initiate the change.

- You must guarantee that your project data is accessible.

The pathnames that you enter into Mentor Graphics applications must direct any process that needs to see your data, including servers, to its location on your network. You are

responsible for guaranteeing accessibility to your data. You must avoid using soft pathnames that are known only on your workstation. Your co-workers cannot use these pathnames to access your data, and you cannot print or plot the data since server processes cannot use them either.

If you use soft pathnames at your site, we recommend that you use location maps to help you ensure that this requirement is met. Location maps make it easier for you to manage and control soft pathnames.

- You must know where to find the master location map and your project location map. You must also establish a location for your private location map.
- You must guarantee that your private location map has exactly the same set of soft prefixes as your project location map.

Your private map should have exactly the same soft prefixes that your project location map has, with no omissions and no additions. If you need to add a soft prefix, your project location map administrator must register it with the master location map administrator and add it to the project map as well.

The best method for guaranteeing this is to create your private location map by copying your project location map and changing or adding only the hard pathnames you need to.

- You should avoid making frequent changes to your private location map.

If you need to make changes to your private location map regularly, you are probably using your map incorrectly. You should reconsider the organization of your project data. Your data's directory structure and its related soft prefixes should remain fairly stable. You should also reconsider the means you use to access the data you are working with, and your intent in selecting among a range of choices.

You should not use soft prefixes to govern the environment of your work area. The need for frequent changes is a danger signal to you that you are applying location maps to problems for which they are not designed.

- You should not use your private location map to isolate your data or make it private. Location maps are not designed to support development activities that require a designer to create a private workspace. You must not use location maps to hide your development changes from others. Mentor Graphics considers this use of location maps to be an abuse of their intent and we warn you that it could lead to severe difficulty. Mentor Graphics does not support an environment in which location maps are employed to create private workspaces.
- You should automate and customize your development environment.

At a minimum, you should set values for the following two shell environment variables:

- MGC_LOCATION_MAP

The value of this shell environment variable is the pathname to your location map file. If you are not using a location map, its value should be “NO_MAP”. Otherwise, its value should be the path to either your project or private location map file.

- MGC_WD

The value of this shell environment variable is the initial value for the MGC working directory, which is known and used by all Mentor Graphics applications. You must be sure that its value is an interoperable absolute pathname. We recommend that you set its value to a hard pathname that maps to a soft prefix in your location map. This allows you to execute the command “cd \$MGC_WD” from your shell, which guarantees that your shell and application working directories are synchronized.

You should set the value of these environment variables automatically in a script file. Depending on your shell, you may need to run the script using the “source” command, or export the variables, so that they appear in the parent shell.

After you set MGC_WD, you might want to synchronize your UNIX working directory and the MGC working directory. This removes any possible confusion that could result if their values are different. If you work in multiple locations within your project, you can create multiple scripts to set both the MGC_WD environment variable and the UNIX working directory, thereby maintaining synchronization of these two values.

Note



You can reset the MGC working directory at any time from inside an application. The MGC_WD environment variable merely provides its initial value when you invoke the application, and synchronization with the UNIX working directory is desirable simply to avoid confusion.

If you log in to different workstations regularly, the values of the MGC_WD and MGC_LOCATION_MAP environment variables may need to be different among those workstations. If this is the case, your script needs logic to decide which workstation you are on, and to set the values accordingly.

If you work on only one project, you can include this script code in your startup file: .profile, .login, or .cshrc. If you work on multiple projects simultaneously, you may want to keep this code in separate initialization scripts for each project. In this case, you should select the project you work on most frequently and execute its initialization script from your startup file. When you need to change project environments, simply run the proper initialization script from the shell command line.

Related Topics

[Master Location Map Administration](#)

[Project Location Map Administration](#)

Server Location Maps

Each shared server process, such as a printer or plotter, must read a location map into memory each time it starts a task.

This provides the server process with access to your company's project data and the libraries you purchase. This also allows you to change the location map to reflect data access updates on your network, without restarting your server. The server picks up all changes from the map the next time it does a job.

Managing location maps for server processes is the responsibility of the master location map administrator. Guidelines for setting up server processes are presented in the list below:

- Do not set values for shell environment variables that have the same names as soft prefixes in the location map, before starting server processes.

In particular, you should not set environment variables from a location map. Because environment variable values override the values in the location map, if your network arrangement changes and you need to change the location map, those changes are invisible to the servers and you need to restart all the servers every time you make a change.
- Use the master location map for your server processes, if possible.

The master location map contains exactly the kind of information that you need to access project and library data on your network. If you can, have the shared servers use the master map.

The two cases in which servers should not use the master location map are listed below:

- If the master map contains pseudo-pathnames to designate locations

Although the master location map normally contains one hard pathname for each soft prefix, which shows how to reach each target from a given example workstation, sometimes these hard pathnames are not available. In this case you use a readable pseudo-pathname that designates the location. In these cases, machine processes cannot use the master map directly.

- If your master map is very large

Very large location maps are sometimes not suitable for servers because of performance impacts. Map size and the order of entries affect the speed of handling pathnames. You should not modify your master location map to facilitate its use by a server process. The master location map must be complete for your network and should be ordered for the convenience of human readability.

- Create one or more server location maps, if your master location map contains pseudo-pathnames or if it is too large.

Create a server location map from the master location map, by copying the master and modifying it to your own needs. Do not create any new soft prefixes. Substitute real hard pathnames for any pseudo-pathnames that exist in the master map.

Since you are creating a server location map, and not using the master location map, you should also do whatever is possible to maximize performance. The DDMS searches a location map sequentially in order to convert pathnames as needed, and always uses the first match it finds. You can optimize search time in the following two ways:

- Place the soft prefixes that are used most frequently ahead of those that are used less often.

You should implement the ordering of soft prefixes in your location map by taking advantage of natural data hierarchies. Place entries for outside libraries in front of the soft prefixes for your projects. These libraries probably take fewer entries and may be accessed more widely.

Then, order your project entries based on their wide-spread use. For example, if you have projects A, B, C, and D at your site, and A, B, and D all use C but not each other, you should put C first in the location map. To order the remaining projects A, B, and D, you may want to consider other factors such as project deadlines and priorities, if pertinent.

- Use the smallest location map possible that permits your server to access all the data that it needs.

When you use only one server map, you do not have an opportunity to decrease the map size. The servers must be able to access all data at your site, and therefore the server location map must use all of the soft prefixes in the master location map.

However, if you have multiple sets of servers, each dedicated to a given work group with its own data access needs, you may be able to partition the master map entries among multiple server maps to decrease their size. You can also adjust the order of entries for each set of servers.

Using multiple server maps can also be useful if you need to install duplicate copies of libraries at different locations around your network. You might do this to minimize network bottlenecks for people accessing those libraries, and/or to provide redundancy in case a disk is lost. The hard pathname entries in the different server maps direct data access to different copies of the libraries, as needed.

The choices you make to optimize server location map performance depend on having suitable information at your disposal. Performance differences are unlikely to be large enough to justify a complex analysis of data access patterns, and you should recognize that the patterns can change over time.

Related Topics

[Master Location Map Administration](#)

[Project Location Map Administration](#)

Location Map Naming Conventions

The selection and use of the soft prefixes and hard pathnames you use in your location map is central to successfully administering your location maps.

Although the following information is intended for the master location map administrator, project and private location map administration practices should be consistent with these guideline where appropriate.

- You should establish a convention for naming soft prefixes, in order to avoid conflicts and confusion. Mentor Graphics suggests the following approach to governing the soft prefix name space. Every soft prefix name should be formed by the ordered concatenation of the following three strings:

- a. company identifier

The company identifier must be unique among companies that use the pathname services. Avoid using the string “MGC_”. Mentor Graphics reserves this company identifier for its own use.

- b. project identifier

The project identifier must be unique within your company. All soft prefixes related to a given project should use the same project identifier. Contact your project location map administrators and have them submit their desired project prefixes for your approval and registration. You must guard carefully against name collisions among projects.

- c. differentiator

The differentiator must be unique within a project. Since each project team creates the soft prefixes it proposes to use, it should be able to avoid name conflicts at this level. However, you should check the names when registering them, as a safeguard. Also, your company may choose to enforce naming standards at that level, as well. If so, you should be the watchdog for maintaining those standards.

It can be helpful to separate the company, project, and differentiator strings with underscores, as shown with the example soft prefix names below:

```
$COMPANY1_PROJXXX_DATA1  
$COMPANY1_PRJXXX_LIB2  
$COMPANY1_XXX_DATA3
```

You can interpret “COMPANY” to mean any character string that identifies your company, and “XXX” to mean any character string that identifies a project. The strings can be of any length of your choice, and can be enforced as a standard length or a variable length.

- You should never use names like “PWD” and “HOME” as soft prefixes in a location map. These names are environment variables that are defined differently, depending upon the current user or situation, and they do not have a consistent meaning across

platforms, shells, or networks. Because of this, you cannot use them to refer conceptually to a single location in the file system, from any workstation, which is the behavior that you want to achieve with a soft prefix.

- You must assign hard pathnames for each soft prefix in a location map to show how to find the data related to the soft prefixes in your file system.

There is never any reason to have more than one hard pathname in an entry of a master location map or of a server location map, since these maps are used only to access existing data, and therefore involve only soft-to-hard pathname conversions.

When assigning hard pathnames to soft prefixes in a location map, you should attempt to do the following items:

- You should never use pathnames that begin with the tilde character (~).

The tilde character has no consistent meaning across platforms, shells, or networks. Therefore, you cannot use it to refer conceptually to a single location in the file system, no matter what machine you look from. For this reason, the DDMS treats the tilde as a literal character in a pathname. It recognizes soft pathnames by looking for a dollar sign (\$) as the first character, and hard pathnames by looking for a slash (/) as the first character. It treats all other pathnames as relative to the MGC working directory.

- You should treat with care any hard pathnames that begin with “//”.

If your network consists of HP Series 400 workstations, you may be using hard pathnames that begin with “//”. These are not valid pathnames on other networks.

It is possible that you also support the previous Mentor Graphics naming conventions that assume the existence of a user directory at the path */user*. Although it is not actually incorrect to place a pathname beginning with “//” in your location map, it slows down pathname processing. If you want to show an equivalence, you should place the equivalence in a comment line of the location map, as shown below:

```
# Project XXX Parts
$COMPANYID_PROJXXX_PARTS
/project/XXX/part
/user/rjones/projXXX/part
# //rjones/local_user/rjones/projXXX/part
```

Related Topics

[Master Location Map Administration](#)

[Project Location Map Administration](#)

Tool Invocation

The Pyxis Project Manager allows you to invoke tools graphically, by selecting and opening objects.

In addition, the Pyxis Project Manager provides you with the ability to create pre- and post-processing scripts that run before and after the tool runs. These scripts are written in AMPLE and can access all of the Pyxis Project Manager design object operations. These scripts do not run when you invoke tools from the shell.

In the Pyxis Project Manager, you can invoke tools in two ways:

1. Data-centered invocation

With data-centered invocation, you specify a design object as input. You navigate to a design object, select it, and “open” it by executing the **Open >** menu item from the navigator’s popup menu. A list of tools appears that can operate on the selected design object. You choose one of those tools and the tool invokes with the selected design object as input.

2. Tool-centered invocation

With tool-centered invocation, you invoke a tool without specifying a design object as input. You double-click a tool icon in the tools window. The tools window displays, as icons, all of the tools that you can invoke from your Pyxis Project Manager session. When you open a tool in the tools window, that tool’s session appears with no data object specified. At this point, most tools either ask you for the pathname of an existing object, or create a new object.

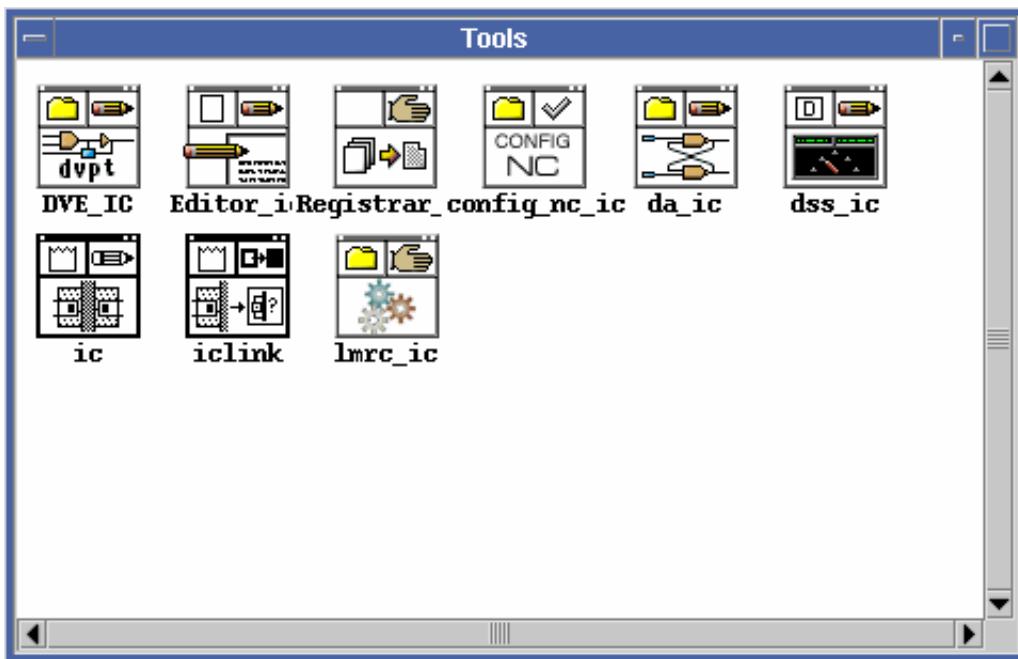
For more information on qualification and termination scripts, refer to the section “[Tool Viewpoints](#)” on page 104 later in this chapter.

The Tools Window 102

The Tools Window

The tools window displays, as icons, the tools that you can invoke from your Pyxis Project Manager session.

Figure 2-14 shows an example of a tools window.

Figure 2-14. The Tools Window

In this context, “invoke” means invoke from the tools window or from a navigator. In other words, when you navigate to a design object, select it, and open it, the Pyxis Project Manager checks the contents of the tools window when it creates the list of compatible tools for that design object.

In general, if a tool does not appear in the tools window, you cannot invoke it from the Pyxis Project Manager. This rule has one exception: if you navigate to a tool viewpoint, which is a special design object that represents the tool, you can open that design object whether or not the tool appears in the tools window. Opening a tool viewpoint invokes the tool that the tool viewpoint represents.

Note

 You do not need to display a tools window in order to open a design object in the navigator. Without a tools window displayed, you can navigate to an existing design object and execute **Open > tool_name** from the navigator popup menu. If *tool_name* is available, it invokes. “Available” means that the tool would have appeared in the tools window, if the tools window was visible.

The tool icons that appear in the tools window represent more than tools; each is a special design object called a *tool viewpoint* that contains AMPLE scripts. When you invoke a tool in the Pyxis Project Manager, these AMPLE scripts also invoke before and after the tool runs. You can edit these scripts to customize the tool pre- and post-invocation behavior.

Tool Viewpoints	104
Pre-tool Qualification and Post-tool Termination Scripts	105

Qualification Scripts	105
Termination Scripts.	107
Toolboxes	108
Changing the Tools Window	108

Tool Viewpoints

A tool viewpoint is a design object that stores the information that the Pyxis Project Manager requires in order to invoke a tool.

The tool viewpoint is a container design object which has a directory and an attribute file. The directory is the default location for your qualification and termination scripts. If you do not want to place your scripts in the tool viewpoint directory, you can add references from the tool viewpoint to your scripts with the Pyxis Project Manager. If a reference to a script exists, the reference takes precedence over the script in the tool viewpoint directory.

Tool viewpoints can have properties and references, and they have an icon. This icon appears in the tools window. As with all design objects, you can copy and delete tool viewpoints in the navigator. Most importantly, like all design objects, tool viewpoints have a type. As a result, you can create many instances of a particular type of tool viewpoint and thus create many customizations of that tool.

You can create custom tool viewpoints in the following two ways:

1. You can copy an existing tool viewpoint and then edit the qualification and termination scripts of your copy.

You can edit a qualification or termination script by performing the following steps:

- a. Navigate to the `$MGC_HOME/toolbox` directory.
- b. Select a tool viewpoint and explore its contents.
- c. Select the file with the `.qual` suffix or `.term` suffix and open it.

2. You can create a new tool viewpoint type by using a Mentor Graphics application called the Pyxis Registrar.

For instructions on creating your own tool viewpoint types, refer to Creating A Tool Viewpoint in the *Pyxis Registrar User's and Reference Manual*.

Related Topics

[The Tools Window](#)

Pre-tool Qualification and Post-tool Termination Scripts

When you invoke a tool from the Pyxis Project Manager, the Pyxis Project Manager executes the qualification script before the tool is invoked and optionally executes the termination script after the tool exits.

These AMPLE scripts accomplish any pre-invocation and post-termination tasks that your tool may require. The following list describes the invocation process:

1. The Pyxis Project Manager calls the tool viewpoint.
2. The tool viewpoint calls the qualification script.
3. The qualification script gathers the tool arguments from you. If you do not supply values for the default arguments (if any), the tool uses the default values in the tool viewpoint.
4. The qualification script calls one of the tool viewpoint's \$invoke functions to invoke the tool's executable file.
5. The tool either brings up a new data object or, if you supplied a pathname to a data object, the tool opens on the existing data object.
6. After the tool exits, the tool viewpoint calls the termination script, if one exists.

You can customize the standard Mentor graphics qualification scripts for your individual use. You can also attach your own termination scripts to Mentor Graphics tools. For example, you might want to add a termination script to update non-Mentor Graphics references. For information on customizing a Mentor Graphics tool, refer to “Customized Mentor Graphics Tools” section in the *Pyxis Registrar User's and Reference Manual* on SupportNet.

To integrate non-Mentor Graphics tools into the Pyxis Project Manager, you must create your own qualification and termination scripts as part of registering the tool. For information on integrating a non-Mentor Graphics tool, refer to “Creating a Tool Type” in the *Pyxis Registrar User's and Reference Manual* on SupportNet.

Related Topics

[The Tools Window](#)

Qualification Scripts

A qualification script is an AMPLE script that you must write to control your tool's pre-invocation behavior and to invoke the tool's executable.

Mentor Graphics applications supply qualification scripts that you can customize. These scripts are located in the tool directories below the `$MGC_HOME/toolbox` directory. An example of a typical tool qualification script is also provided as part of “Example Encapsulation” in the *Pyxis Registrar User's and Reference Manual* on SupportNet.

The name of the qualification script must be the tool viewpoint name followed by the “.qual” suffix. For example, the name of the qualification script for a tool viewpoint named “custom_font_editor” must be “custom_font_editor.qual”.

When writing your qualification script, you can use any of the tool viewpoint functions and the functions available in the Pyxis Project Manager session window.

You can also use the AMPL variable \$qual_script_scope in your qualification script. The \$qual_script_scope variable returns the scope name of the qualification script. Because the qualification script scope is temporary, this variable is useful when you want to share data between qualification and termination scripts.

For more information on sharing data between qualification and termination scripts, refer to “Sharing Data Between Qualification Scripts and Termination Scripts” in the *Pyxis Registrar User's and Reference Manual* on SupportNet.

Table 2-6 lists the tool viewpoint functions.

Table 2-6. Tool Viewpoint Functions

Function	Description
\$get_object_pathname()	Returns the absolute pathname of the specified design object during data-centered tool invocation, or the absolute pathname of the active tool viewpoint during tool-centered tool invocation.
\$get_object_type()	Returns the type of the specified design object during data-centered tool invocation, or the type of the active tool viewpoint during tool-centered tool invocation.
\$get_object_version()	Returns the version number of the specified design object during data-centered tool invocation, or the version number of the active tool viewpoint during tool-centered tool invocation.
\$get_tool_pathname()	Returns the absolute pathname of the active tool viewpoint during data-centered or tool-centered tool invocation.
\$get_tool_script()	Returns the absolute pathname of the active tool viewpoint's qualification or termination script.
\$get_tool_type()	Returns the type of the active tool viewpoint during data-centered or tool-centered tool invocation.
\$invoke_bgd_tool()	Invokes an executable file as a background process.
\$invoke_tool()	Opens a new shell window and invokes an executable file.
\$setup_invoke_tool()	Sets the active tool viewpoint.

You can write your qualification script to perform tasks like collecting input arguments from the user; implementing process control; and, when used in conjunction with a termination script, converting your data to and from non-Mentor Graphics formats to the Mentor Graphics format.

In general, the qualification script should contain the following components:

- AMPLE functions to create tool invocation dialog boxes.
- An AMPLE function that controls tool invocation based on the design object selected.

When you are writing your qualification script, you can use a case statement to determine the actions to be done according to the type of the design object selected. The case statement needs to contain one statement for each data type listed in the input_data property on the tool type, a statement for the tool type itself, and a default statement that returns an error if the type you have selected is not found in the other statements.

For tool-centered invocations, you typically use the qualification script to collect the tool invocation arguments and switches from the user. For data-centered invocations, you typically use the qualification script to get the pathname to the data object and also to gather the tool invocation arguments and switches from the user.

Calling one of the tool viewpoint \$invoke functions in the qualification script invokes the tool. If the qualification script does not call \$invoke_tool() or \$invoke_bgd_tool(), the executable for the tool is not invoked and the termination script is not executed.

As previously mentioned, the qualification script should gather the tool invocation arguments. However, you can use the qualification script to accomplish any task you want. For example, aside from gathering the tool invocation arguments, you can have the qualification script validate the user's argument choices or have the script check for prerequisite steps before tool invocation.

Related Topics

[The Tools Window](#)

Termination Scripts

The termination script is an optional AMPLE script that performs any post-processing of newly created or modified design objects.

This script, if one exists, is executed in the Pyxis Project Manager after the tool executable completes execution. An example of a typical tool termination script is provided as part of the example in “Example Encapsulation” in the *Pyxis Registrar User’s and Reference Manual* on SupportNet. By default, Mentor Graphics tool viewpoints do not have a termination script.

The tool viewpoint functions \$get_object_pathname() and \$get_object_type() return the same information as was available to the associated qualification script at the time of the tool invocation.

In addition, the following two AMPLE variables are available to you in a termination script:

- `$term_script_scope`

This AMPLE variable returns the scope name of the termination script.

- `$tool_termination_status`

This AMPLE variable returns the termination status of a tool. This termination status is 16 bits that differentiate between stopped and terminated child processes and, in the later case, identify the cause of termination.

For example, if a tool is terminated as the result of a signal, the high-order 8 bits of status would be zero and the low-order 8 bits would contain the number of the signal that caused the termination. In addition, if a process dump had occurred, the low-order seventh bit would be set.

Related Topics

[\\$get_object_pathname\(\)](#)

[\\$get_object_type\(\)](#)

Toolboxes

A toolbox is a directory that holds tool viewpoints.

For example, the standard Mentor Graphics tools may be in one toolbox, your customized versions of Mentor Graphics tools in another, and your third-party tools in another. To display the tools that you can invoke, the Pyxis Project Manager maintains a toolbox search path to perform an ordered search through multiple toolboxes.

The Pyxis Project Manager uses toolboxes in conjunction with the toolbox search path to locate tools for invocation and to display tool icons in the tools window. For each user, the Pyxis Project Manager maintains a separate toolbox search path. The search path consists of an ordered list of toolboxes (directories). If several toolboxes in the search path have tool viewpoints for a single tool, the Pyxis Project Manager executes the first tool viewpoint it encounters in the search path.

For information on creating a toolbox, refer to *Creating a Toolbox* in the *Pyxis Registrar User's and Reference Manual* on SupportNet.

Related Topics

[The Tools Window](#)

Changing the Tools Window

The tools window displays all of the tools that you can invoke from the Pyxis Project Manager.

As a result, if you want to make a new tool accessible in your Pyxis Project Manager session, you must make its icon visible in the tools window. This new tool could be one of your own customizations of an existing tool viewpoint, or it could be a tool that you did not previously want to invoke from the Pyxis Project Manager.

To change which tools appear in the tools window, you must change your *toolbox search path*. The toolbox search path is an ordered list of directories or *toolboxes*. Each toolbox contains tool viewpoints. The Pyxis Project Manager searches the toolboxes specified by the search path, looking for tool viewpoints to display in the tools window. This search occurs when you do one of the following:

- Invoke the Pyxis Project Manager
- Change the toolbox search path
- Update the tools window

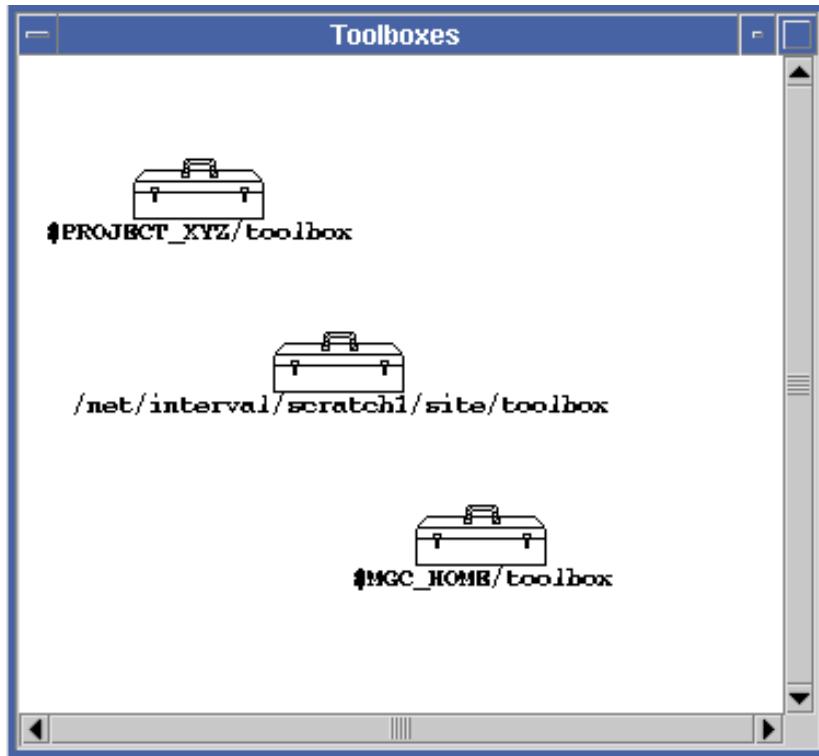
If a path in the toolbox search path cannot be found, the following error message displays:

Cannot find toolbox at <path>. Please verify your toolbox search path located at
\$HOME/mgc/startup/dmgr_toolbox_path.

To view your toolbox search path, you execute the View Toolboxes command from the tools window's popup menu. The tools window then switches to toolbox mode and displays your toolbox search path graphically, as a row of toolboxes. Figure 2-15 shows a tools window in toolbox mode. In this figure, the soft prefix *\$PROJECT_XYZ* represents the location of the project directory for the XYZ project, and the pathname */net/interval/scratch1/site* represents an arbitrary location on another workstation where a toolbox resides.

When searching for tools to display in a tools window, the Pyxis Project Manager searches the toolboxes, as shown in Figure 2-15, from the upper left toolbox to the lower right toolbox. In the example, the Pyxis Project Manager would search the toolboxes in the following order: *\$PROJECT_XYZ/toolbox*, */net/interval/scratch1/site/toolbox*, and *\$MGC_HOME/toolbox*. Note the name *\$MGC_HOME/toolbox*. This toolbox is the directory in which the standard Mentor Graphics tools reside.

Figure 2-15. Example of a Toolbox Window



The toolbox search path has two purposes:

- It allows you to add tools (tool viewpoints) to the tools window by adding toolboxes to the search path. When you add toolboxes to the search path, the Pyxis Project Manager looks in the newly added toolboxes and displays, and makes accessible any tools found inside.
- It allows you to display one tool viewpoint instead of another. Remember, tool viewpoints are design objects that, like all design objects, have a particular name. This name appears under the tool icon in the tools window, and at a given time, the tools window displays exactly one tool viewpoint of a given name. If you have two tool viewpoints, each with the same name, you can make one accessible and hide the other by changing the toolbox search path.

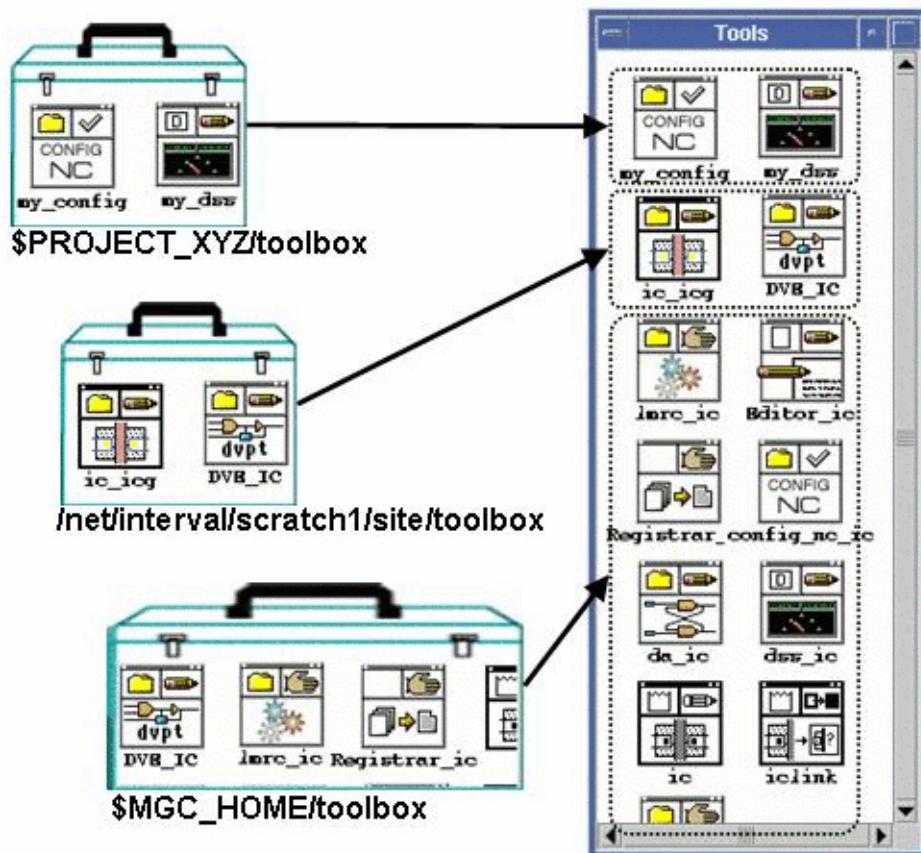
The following example clearly demonstrates these two purposes. Assume that a user defines and saves his toolbox search path as shown in Figure 2-16. When populating the tools window, the Pyxis Project Manager searches the toolboxes in the following order:

1. \$PROJECT_XYZ/toolbox
2. /net/interval/scratch1/site/toolbox
3. \$MGC_HOME/toolbox

When the Pyxis Project Manager finds a tool viewpoint whose name it has not seen before during the search, it displays that tool viewpoint in the tools window. When displayed in the toolbox window, the tool viewpoint's type is shown graphically with an icon, and the name is shown beneath the icon.

In this example, the tool viewpoint named *my_dss* that appears in the tools window is the one that the Pyxis Project Manager found in *\$PROJECT_XYZ/toolbox*. The toolboxes are searched for unique names, not types. When the Pyxis Project Manager checked toolbox */net/interval/scratch1/site/toolbox* for tool viewpoints, it found only one uniquely named tool viewpoint: *build_ug*. Although the two *my_dss* tool viewpoints are of different types (as indicated by their different icons), they share the same name; as a result, only one icon, the first one encountered, is displayed in the tools window. To make the *my_dss* tool viewpoint in */net/interval/scratch1/site/toolbox* accessible, that toolbox must be moved above the *\$PROJECT_XYZ/toolbox* in the toolbox search path.

Figure 2-16. Toolbox Search Path



Note

To save the modified toolbox search path, you must execute either the toolbox popup menu item **Save** or the pulldown menu item **Edit > Save**.

You can change the toolbox search path graphically with select-and-drag techniques or with commands entered in the popup command line. To save this change, however, you must explicitly save the toolbox search path by using one of the popup or pulldown menu items **Save** or **Edit > Save**, or by executing the `$save_toolbox_search_path()` function.

You can also modify one of the Pyxis Project Manager startup scripts to specify a particular toolbox search path. When you invoke the Pyxis Project Manager, it uses that toolbox search path to populate the tools window.

For information on site-specific and workstation-specific startup files, refer to the manual, *Customizing the Pyxis Common User Interface* available on SupportNet.

Related Topics

[The Tools Window](#)

[Changing the Toolbox Search Path](#)

[AMPLE for Pyxis User's Manual](#)

Startup Files

The Pyxis Project Manager supports the standard Mentor Graphics startup file mechanisms and also supports saving session setup information and the toolbox search path in additional startup files.

As with all Mentor Graphics applications, the Pyxis Project Manager searches the startup directories for startup files, when it invokes. The directories are searched in the order presented below, and each startup file's execution can potentially override the effect of the previous startup file's execution.

1. `$MGC_HOME/shared/etc/cust/startup`

This directory contains site-specific startup files. These files are customized to invoke applications for your workplace. These startup files contain specific userware, which is common to all users at your site. For example, the default background color is changed to "cyan" for all Pyxis Project Manager windows for your site, by placing the following command in `$MGC_HOME/shared/etc/cust/startup/dmgr.startup`:

```
$set_bgd_color("cyan");
```

2. `$MGC_HOME/etc/cust/startup`

This directory contains project-specific startup files. These files are customized to invoke applications for your project. For example, to change the default window tiling for your project's use of Pyxis Project Manager, place the following line in `$MGC_HOME/etc/cust/startup/dmgr.startup`:

```
$set_default_method(@quad);
```

For information on using location maps, refer to the section “[Design Management with Location Maps](#)” on page 71 in this chapter.

3. `your_home/mgc/startup`

This directory contains user-specific startup files. You can customize these startup files to fit your personal working environment. For example, to change the active window highlighting for your own Pyxis Project Manager session to “lightblue,” place the following line in `your_home/mgc/startup/dmgr.startup`:

```
$set_active_color("lightblue");
```

The user-specific `dmgr.startup` startup file allows you to customize various items, such as key definitions, application-specific commands, menus, prompt bars, dialog boxes, and stroke definitions. You can also open windows at the time of invocation. Additionally, any of the options in the **MGC** pulldown menu are available.

Table 2-7 presents functions that you can use in your `dmgr.startup` file. These are Pyxis Project Manager-specific functions that are available in the session window menus. All Common User Interface functions that exist beneath the session scope are also available.

Table 2-7. Pyxis Project Manager Session Window Functions

Function	Description
<code>\$open_configuration_window()</code>	Opens a new or existing configuration object.
<code>\$open_navigator()</code>	Opens a navigator.
<code>\$open_session_monitor()</code>	Opens the session monitor.
<code>\$open_tools_window()</code>	Opens a tools window.
<code>\$open_trash_window()</code>	Opens a trash window.
<code>\$open_types_window()</code>	Opens a types window.
<code>\$set_toolbox_search_path()</code>	Sets the order of the toolbox search path.
<code>\$setup_default_editor()</code>	Specifies the default ascii editor.
<code>\$setup_filter_all()</code>	Specifies navigator filters.
<code>\$setup_iconic_window_layout()</code>	Specifies the graphical layout of iconic windows.
<code>\$setup_startup_windows()</code>	Specifies whether a tools window or navigator window is opened at invocation.
<code>\$setup_monitor()</code>	Specifies configuration monitoring characteristics.
<code>\$setup_session_defaults()</code>	Specifies session-wide characteristics of the Pyxis Project Manager graphical interface.

In addition to the *dmgr.startup* file, which is commonly referred to as either the *application* or *custom startup file*, two other startup files can exist in the *your_home/mgc/startup* directory. These two files are *dmgr_default.startup*, which is referred to as the *default startup file* and *dmgr_toolbox_path.startup*, which is referred to as the *toolbox search path startup file*. These files have specific purposes, though, like all startup files, they can contain any valid AMPLE function calls.

The default startup file and the application startup file are mutually exclusive, in that only one of them executes, even if both exist. The application startup file always takes precedence. If the application startup file exists, it is executed and the default startup file is ignored. If the application startup file does not exist *and* no other site-specific or project-specific startup files exist, the default startup file is executed.

The toolbox search path startup file is the first startup file to be executed, if it exists. It is executed even before the site-specific and project-specific startup files. In other words, it has the lowest precedence, and the actions it performs can be overridden by the actions of any of the startup files which execute after it.

The Default Startup File	114
The Toolbox Search Path Startup File	115
Startup File Search Order	116

The Default Startup File

The purpose of the default startup file is to allow various desired session settings to be preserved between invocations of the Pyxis Project Manager.

The first time that you execute one of the menu items available through the session pulldown menu **Setup**, you are prompted:

Save setup changes in default startup file?

If you click on the **No** button, the settings you have just specified and any subsequent new settings that you specify in that session are not saved.

If you click on the **Yes** button, the Pyxis Project Manager checks for the existence of a default startup file. If a default startup file exists, you are prompted:

Default startup file exists. Overwrite it?

If you respond **No**, your setup changes for the session and any subsequent new settings that you specify in that session are not saved. If you respond **Yes**, the Pyxis Project Manager checks for the existence of any of the three (site, project, custom) startup files. If any of them exist, you are prompted:

Site, project, or custom startup exists and will be used instead of default startup.

Save default startup anyway?

If you respond **No**, your setup changes in that session and any subsequent new settings that you specify in that session are discarded. If you respond **Yes**, the settings you have just specified are saved to the default startup file *dmgr_default.startup*. If the default startup does not already exist, the Pyxis Project Manager creates it in the following directory: *your_home/mgc/startup*

Remember that because the default startup file only executes if the application startup file *dmgr.startup* does not also exist *and* no other startup files exist, file setup instructions in your default startup file may not always be executed during Pyxis Project Manager invocation.

Using the default startup file, in the absence of a user-specific startup file, allows you to change default settings without worrying about which startup file to modify.

Be aware that the **Setup** > pulldown menu is different from that supplied by the Common User Interface, which is the **MGC** > **Setup** > menu. The **Setup** > menu allows you to specify Pyxis Project Manager-specific session values and save them in your default startup file. When you change session settings using the **MGC** > **Setup** > menu, your settings are not saved.

Note



You can use the **Setup** > menu to automatically create the default file *dmgr_default.startup*, containing the commands that specify the default editor, icon layout, startup windows, session defaults, navigator filters, and configuration monitoring characteristics. You can then “cut and paste” the contents of that file into your customized application startup file *dmgr.startup* and add to the existing commands. Alternatively, you can simply rename *dmgr_default.startup* and use it as a starting point for a *dmgr.startup* file.

Related Topics

[Change Your Session Setup](#)

The Toolbox Search Path Startup File

The toolbox search path determines the order in which toolboxes are searched.

The search path can be changed interactively by using the click and drag capabilities in the toolbox window. At any time during a Pyxis Project Manager session the toolbox search path can be saved. When this is done, the current search path is written to the file:

your_home/mgc/startup/dmgr_toolbox_path.startup

This file is executed automatically during the next Pyxis Project Manager invocation, causing the toolbox search path from the previous session to be preserved. Alternatively, the toolbox

search path can be explicitly set in either the application startup file or the default startup file. If the toolbox search path is not explicitly specified in one of the three possible startup files in the directory `your_home/mgc/startup`, the default toolbox search path is automatically set to `$MGC_HOME/toolbox`.

The `dmgr_toolbox_path.startup` file is unique to the Pyxis Project Manager application. The toolbox search path startup file can contain commands other than `$set_toolbox_search_path()`. However, because the file is overwritten whenever the current search path is saved, any additional commands contained in it are lost.

If you are using a location map, the `$MGC_HOME/toolbox` directory, as well as your user directory, must be entries in the location map.

Related Topics

[Changing the Toolbox Search Path](#)

[Design Management with Location Maps](#)

Startup File Search Order

The Pyxis Project Manager executes startup files sequentially at invocation.

1. The Pyxis Project Manager always reads the toolbox search path startup file `your_home/mgc/startup/dmgr_toolbox_path.startup` first, when it invokes.
2. If a site-specific startup file exists, the Pyxis Project Manager executes it next.
3. If a project-specific startup file exists, it is executed after the site-specific startup file.
4. The Pyxis Project Manager then executes your application startup file `your_home/mgc/startup/dmgr.startup` if it exists. If it does not exist and no other site or project startup files exist, the Pyxis Project Manager executes `your_home/mgc/startup/dmgr_default.startup`.
5. If no site, project, or user-specific startup files exist, the Pyxis Project Manager provides a default setup which specifies to open a tools and a navigator window.

If you are not familiar with AMPLE, see your system administrator for assistance in modifying these startup files.

Related Topics

[Pyxis Common User Interface User's Manual](#)

[AMPLE for Pyxis User's Manual](#)

Configuration Management Concepts

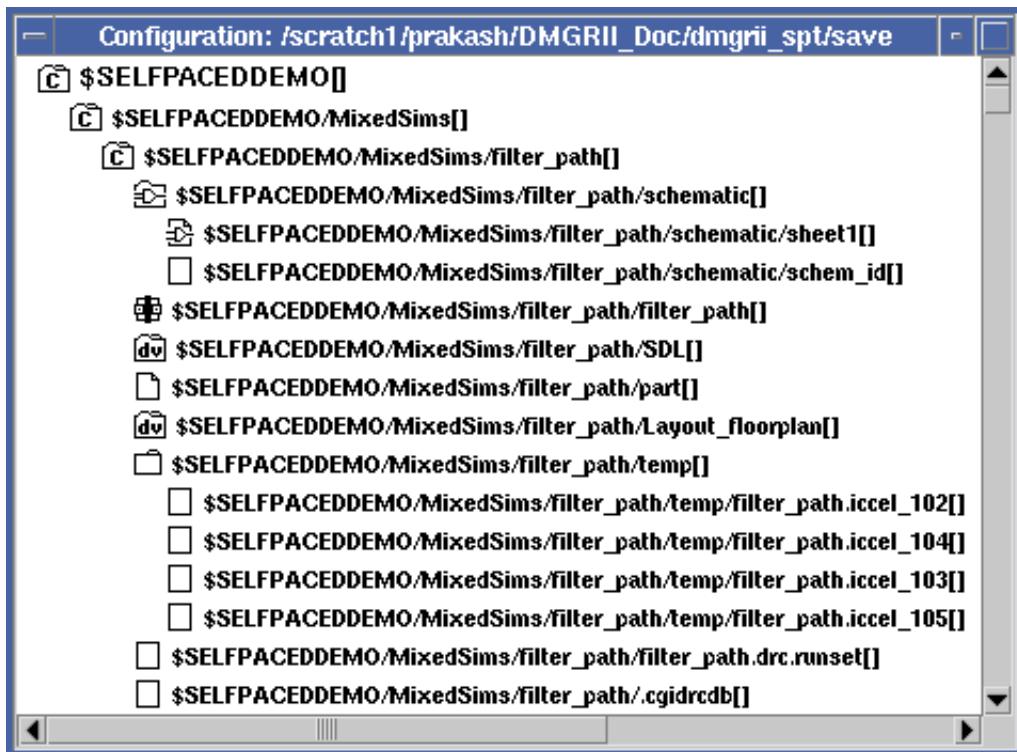
Design object references allow you to distribute your design across many locations in the file system.

For example, your design may be a schematic that references parts in common libraries on another workstation. In addition, you may have used the Pyxis Project Manager to create a reference from your schematic to the documentation that describes it, and that documentation may reside in a different directory than the schematic.

Using references to distribute designs is a powerful feature, but it makes operating on the entire design as a single unit very difficult. For example, you might want to copy the entire design to a different workstation. To do that, you must locate and select all of the elements in the design before doing the copy. Even if you know where each design element resides, and you probably do not, navigating to and selecting each one prior to a single operation would be difficult.

To solve this problem, the Pyxis Project Manager provides *design data configuration management*. In the Pyxis Project Manager, design data configuration management consists of two items:

1. **Configuration window.** This window, shown in Figure 2-17, provides a convenient, automated way to gather all elements of a distributed design into one place. The gathered elements, which are a set of related design objects, are called a *configuration*. After you create a configuration in this window, you can perform several powerful operations on the entire configuration, including delete, copy, and release (protected copy).

Figure 2-17. Configuration Window

2. **Configuration object.** This object, shown in Figure 2-18 with its large and small icon, is a special design object that corresponds to the configuration window.

Figure 2-18. Large and Small Configuration Object Icons

The configuration object records the exact contents of the configuration after you create it. This object does not *contain* the elements of a design; it is not a directory. Instead, it simply records which design objects are part of a configuration. By navigating to and opening the configuration object, you can recreate a configuration after you have closed the configuration window.

Creating a Configuration	118
Managing Configurations in Batch Mode	122
Configuration Monitoring	122
Configuration Operations	123

Creating a Configuration

A configuration is a set of design objects that are related by *containment* or by references.

A simple example of a configuration is a directory. The top-level directory and all design objects in the tree beneath it form a configuration. To operate on the objects as a single unit, you simply operate on the top-level directory.

A directory is a simple but useful configuration, and you do not need the help of the configuration window to create and to organize a directory. However, not all designs are completely contained in directories, and a directory may contain items that you do not want to include in the configuration. To handle these cases, you need to use the configuration window to create a configuration.

To create a configuration in the Pyxis Project Manager, you need to understand the following five concepts:

1. Opening a configuration window

The configuration window displays the configuration as you create and change it. If you save the configuration, a configuration design object records the information that is displayed in this window. After you build a configuration and are satisfied with it, you can operate on it, using operations from the configuration window's menus.

2. Adding primary entries

In this step, you manually add items to the configuration window. You do this by executing a command or by dragging objects into the configuration window from a navigator. These items are called *primary entries* because they are usually a major part of a design, such as a design directory. The remaining entries of the configuration, the *secondary entries*, are brought in automatically by virtue of their relationship with the primary entries.

This step provides you with the first level of specifying your configuration. Even if you do not specify any complicated build rules for gathering more entries in the following step, you can still create a useful configuration simply by adding entries manually. For example, you can bring a directory and its contents together with a file from a completely different location. When you copy this configuration, the file and the directory are siblings in the target directory. Thus, the file and the directory are treated as a single unit.

Strictly speaking, configuration entries, both primary and secondary, are not complete design objects. Rather, they are each a single version of a design object. This distinction allows you to capture previous versions of design objects in the configuration. Taken together, these previous design object versions and their reference relationships define a previous version of an entire design. Thus, the configuration window lets you capture a previous version of a whole design.

3. Setting build rules

In this step, you select primary entries and specify a set of *build rules* for those entries. When you build the configuration, the Pyxis Project Manager uses these rules to search

for secondary entries to add to the configuration. The secondary entries are related to the primary entries by containment or by reference.

When you specify a set of build rules with *multiple* primary entries selected, the build rules for all of those selected primary entries are set according to the values you specify. Although the build rules are the same for each of the selected entries, each entry has its own set of build rules.

The default build rules specify whether to include everything contained in or referenced by the primary entry. Here is a brief description of the build rules that you can specify:

- **Reference traversal.** When you specify the default reference traversal, the command for building the configuration, Build Configuration, traverses the entire reference network of the primary entry. In other words, if primary entry A references directory B, which references file C, then Build Configuration potentially includes A, B, the contents of B (including any objects that those contents reference, their references, and so on), and C.

The key phrase is “potentially includes.” You can exclude potential entries from the configuration by using filters.

- **Other filters.** You can use filters to include or exclude certain design objects that the Build Configuration command potentially includes while traversing containers and references. You can choose any or all of the following filter categories:
 - Object pathname: Filters design objects based on patterns in their pathnames.
 - Object type: Filters design objects based on their type.
 - Object property: Filters design objects based on properties that they hold.
 - Reference property: Filters design objects based on properties that are held by the reference that points to them.

Using reference traversal and filters, you can build a very complex and precise configuration of related design objects.

4. Building the configuration

When you build the configuration, the Build Configuration command starts with each primary entry, traverses its contents and possibly its references, filters out certain objects, and adds secondary entries to the configuration. The status of the Build Configuration command, including progress information such as failures, errors, and warnings, is reported and displayed in the configuration monitor window during the course of the operation.

If your session setup values specify to show the monitor window, the session monitor window is opened when you execute this function. When the build is complete, the monitor window remains visible until you remove it.

You execute the monitor window's popup menu item **Hide Monitor** to hide the monitor window. This command hides the monitor window and re-displays the updated configuration. At this point, you can copy, delete, release, or perform other operations on the configuration, and it is treated as a single unit. You can revisit the build operation's status report in the configuration monitor window, by executing the configuration's popup menu item **Show Monitor**.

Because the time needed to build a seemingly simple configuration can be deceiving, if the objects in the configuration have a very large containment and/or reference network, you have the option of halting the execution of the build operation. You can interrupt the build operation, by pressing the Ctrl and Backslash keys together.

When you halt the build operation, a dialog box is displayed prompting you if the operation should abort or continue. If you choose to abort the operation, the standard clean-up procedure is performed.

5. Saving the configuration

After building the configuration, you can save it to a configuration object. This object records the exact contents of the configuration and allows you to recreate it later, without manually adding items, specifying build rules, and so on. Instead, you simply open the saved configuration object and the configuration window appears, displaying the contents of the configuration.

When you save the configuration, the resulting *configuration object* is automatically added to the configuration. A configuration object is a special type of design object that specifies how a configuration is built. This feature is especially useful when you release a configuration to another location. Because the configuration object is now part of the configuration, it is included in the release. As a result, to view the contents of the release in the target directory, you need only open the configuration object. In addition to recording the contents of a configuration, a released configuration object also records important information about the release, such as the time and owner of the release and the source location of the released objects.

After you create configurations interactively a few times and understand these concepts, you can use batch mode scripts, using Pyxis Project Manager AMPLE functions, to create and manage your configuration.

Related Topics

[Opening a New Configuration](#)
[Adding a Primary Entry](#)
[Specifying the Build Rules](#)
[Saving a Configuration](#)

[Change Your Session Setup](#)
[Build a Configuration](#)
[\\$build_configuration\(\)](#)

Managing Configurations in Batch Mode

You use configuration management functions and design object toolkit functions to supplement the capabilities of the Pyxis Project Manager configuration window by creating scripts that perform configuration operations in batch mode.

The configuration management toolkit functions support an active configuration model. That is, if a configuration toolkit command is issued through a configuration window, the command is applied to the active configuration window's configuration; if a configuration toolkit command is issued when a window other than the configuration window is active, the command is applied to the session configuration. The functions `$$open_configuration` and `$$create_configuration`, which are used to initialize the session configuration, and the function `$$close_configuration`, which is used to close the session configuration, target the session configuration, and therefore, are not available from the configuration window.

Related Topics

[\\$\\$open_configuration\(\)](#)

[\\$\\$create_configuration\(\)](#)

[\\$\\$close_configuration\(\)](#)

Configuration Monitoring

The Pyxis Project Manager provides real-time status monitoring for configuration operations. Configuration monitoring supports an active configuration model.

If an interactive configuration command is issued from a configuration window, status is reported to the configuration monitor window. If a configuration toolkit command is issued through a configuration window, status is reported to the configuration monitor window. If a configuration toolkit command is issued when a window other than the configuration window is active, the command is applied to the session monitor window.

You view the configuration monitor window by executing the **Show Monitor** menu item on the configuration window's popup menu. You view the session monitor window by executing the **Windows > Open Session Monitor** pulldown menu item. Both the configuration monitor window and the session monitor window are usually referred to as the monitor window, since they differ only in that they target different configurations.

The Pyxis Project Manager allows you to customize the behavior of configuration monitoring behavior. You can specify to filter out classes of errors, warnings, and informational messages from monitor reports, whether status information is output to the monitor window or to the transcript window, whether monitor windows are visible or hidden, and you can specify the verbosity of monitor reports. By default, all errors, warnings, failures, and informational messages are monitored and reported, the monitor status is output to the monitor window, the session monitor window is hidden from view, and configuration monitoring verbosity is relatively terse.

The verbose option allows you to specify additional information and context in configuration monitor reports, or to specify that a small subset of the available status information is reported. For example, when the `$$change_configuration_references()` function is executed with verbosity set to verbose, each individual reference is monitored and the monitor report includes the pathname of each reference before the change and after the change operation, along with the total number of references that were changed. When the `$$change_configuration_references()` function is executed with verbosity set to terse, only the total number of changed references is reported.

Related Topics

[Change Your Session Setup](#)

[Function Summary](#)

Configuration Operations

After you create a configuration, you can operate on it with the commands that are available in the configuration window.

These commands operate on every design object version in the configuration, in a single operation.

For detailed instructions on how to perform configuration operations, refer to the section “[Managing Designs](#)” on page 238 in Chapter 3.

Copying a Configuration	123
Releasing a Configuration	125
Freezing a Configuration	128
Deleting a Configuration	130
Changing the References of a Configuration	130
Locking a Configuration	131

Copying a Configuration

In a single operation, you can copy each design object version in the configuration to a single target directory.

Note the following features of the configuration copy operation:

- Because every configuration entry is only a single version of a design object, when you copy the configuration, the Pyxis Project Manager performs a “copy version” on each entry. That is, the version number of the copied object is renumbered in the target directory. For example, if the configuration contains only version number 8 of object A, then in the target directory, the copy of A has only a single version: version 1. If the configuration contains two versions of object A, versions 6 and 8, then in the target directory, these copies are renumbered and compacted to versions 1 and 2.

- In the destination directory, containment relationships are preserved. For example, if entry *A* contains entry *B*, in the target directory, *A* still contains *B*.
- As with the Copy Object command, when you copy a configuration, the references between the selected design objects are updated to reflect the new location. Because every object in the configuration is “selected” (that is, each one is copied by the configuration operation), the references between every configuration entry are updated to reflect the new location.
- You can include, in the configuration, the configuration object that describes the configuration. As a result, that object is copied to the target directory, and you can recreate or change the configuration in the target directory. When you save an unnamed configuration, the Pyxis Project Manager automatically adds that configuration's design object to the configuration.

Including the configuration object in the configuration is also important because the Copy Configuration command places helpful record keeping information on the references of that object. The configuration object uses references to keep track of exactly which objects are part of the configuration. The configuration object has one reference to each entry.

When you copy a configuration, the Pyxis Project Manager places the following two properties on each reference between the copied configuration object and the copied configuration entries:

- A property named **from_path**, the value of which is the pathname of the source design object.
- A property named **from_version**, the value of which is the version number of the source design object.
- When you copy a configuration, the status of the operation is reported and displayed in real-time in the monitor window.

If your session setup values specify to show the monitor window, the monitor window is opened when you execute this function.

- The copy configuration command can be interrupted. When you copy a configuration, all objects that it contains are also copied. The time needed to completely copy a seemingly simple configuration can be deceiving, if the configuration has a very large containment and/or reference network. In a case like this, you can halt the operation's execution, by pressing the Ctrl and Backslash keys together.

When you halt the copy operation, a dialog box is displayed prompting you if the operation should abort or continue. If you choose to abort the operation, the standard clean-up procedure is performed.

Related Topics

[Getting Information about a Copy](#)
[Change Your Session Setup](#)

[Copy Design Object](#)
[\\$copy_configuration\(\)](#)

Releasing a Configuration

A configuration release creates a protected copy of a configuration, from the configuration.

The Release Configuration command has these major features:

- The Release Configuration command is performed from the configuration window or from the toolkit scope.
- As with Copy Configuration, with the Release Configuration command, in the target directory, containment relationships are preserved, references are updated, and versions are renumbered starting at number 1.
- Unlike the Copy Configuration command, the Release Configuration command assigns the “released” attribute to every versioned object in the release directory and the Pyxis Project Manager does not allow you to create new versions of these released design objects. If you want to edit these design objects, you must copy the released configuration to another directory.

Remember that the “released” attribute only prevents you from opening and editing *versioned* objects in the release directory. It does not prevent you from opening an unversioned design object, or from deleting or moving *any* released design object. To preserve the integrity of your release, avoid these operations in the release directory.

- You can choose to release parts of the configuration to different destinations by setting different target paths for entries in the configuration.
- As part of the release operation, you can choose to have the references on the design object updated to reflect the new location.
- When you release the same configuration to the same target directory multiple times, the Release Configuration command “layers” the versioned design objects, creating a new version of the configuration for each release. For versioned design objects only, releasing does not overwrite old data with new data. If a versioned design object is unchanged since the last release, the Release Configuration command does not copy the configuration to the target directory. If the configuration has changed, the Release Configuration command merges the new version with the existing design object in the target directory, thus creating a new version of that object.

If you save the configuration, and as a result include the configuration object that describes the release in the configuration, at the target directory, you can access particular versions of these layered releases. Because the configuration object is versioned, when you release to the target directory, the configuration object is layered

like other versioned design objects. Each version of the configuration object describes a particular set of design object versions. In other words, each version of the configuration object describes a particular version of an entire design. An example of a layered design is shown in Figure 2-19.

In addition to releasing the design object versions and the configuration object, this function also releases the in-memory location map to the destination directory.

Note

 Release layering only occurs for versioned design objects. The data in unversioned objects is overwritten with each repeated release to the same directory. When unversioned containers are released, not only the container, but all the objects it contains, are overwritten. The one exception to this rule applies to containers of type “mhc_container” which are ordinary directories. Containers of this type are handled as a special case, to allow designs contained in ordinary directories to layer during a release. With the exception of this special case, **containers should always be versioned**, to allow their contents to be layered during the release process.

You need to ensure that important data are contained in “versioned” design objects, if you want to layer your releases. An alternative to layering is to choose a new directory for each release.

- The release operation reports and displays real-time status in the monitor window.
If your session setup values specify to show the monitor window, the monitor window is opened when you execute this function. When the operation is complete, the monitor window remains visible until you remove it, by executing the monitor window's popup menu item **Hide Monitor**.
• The release operation can be interrupted. The time needed to completely release a seemingly simple configuration can be deceiving, if the objects in the configuration have a very large containment and/or reference network. In a case like this, you can halt the execution of the release operation, by pressing the Ctrl and Backslash keys together.
When you halt the release operation, a dialog box is displayed prompting you about whether the operation should abort or continue. If you choose to abort the operation, the standard clean-up procedure is performed.
- Like the Copy Configuration command, Release Configuration places important record keeping information on the released configuration object. It also places information about the release on the source configuration object.

You can use the Release Configuration command to create successive revisions of the same design by repeatedly releasing a configuration to the same target directory. To identify each revision, you use the Annotation field in the Release Configuration dialog box.

Procedure

To extract Revision 2 of the design as shown in the example, you would perform the following steps:

1. Navigate to the release directory.
2. Select the configuration object that describes the release.

This object is included and modified with each release.

3. Execute **Report > Show Versions** from the navigator's popup menu.

A version window appears, showing versions 1 and 2 of the configuration object. These versions correspond to Revisions 1 and 2 of the design.

4. Select version 2 of the configuration object and execute the **Report Version Info** menu item from the version window's popup menu.

An information window appears, showing information about version 2 of the configuration object. Beneath the heading “Version Properties”, the text is “release comments: Revision 2”.

This step verifies that Revision 2 of the design is defined by version 2 of the configuration object.

5. Close the information window and return to the version window.
6. In the version window, double-click on version 2 of the configuration object.

A configuration window appears, displaying the configuration. Although version 2 is the most recent version of the configuration object, the configuration window treats all items in the version window as previous versions. Therefore, version 2 is protected, and you cannot edit in this window. Non-destructive operations such as copy, are available.

7. Execute **Global Operations > Copy** from the configuration's popup menu.

The Copy Configuration dialog box appears.

8. At the Destination Directory prompt, enter a directory other than the release directory, and execute the dialog box.

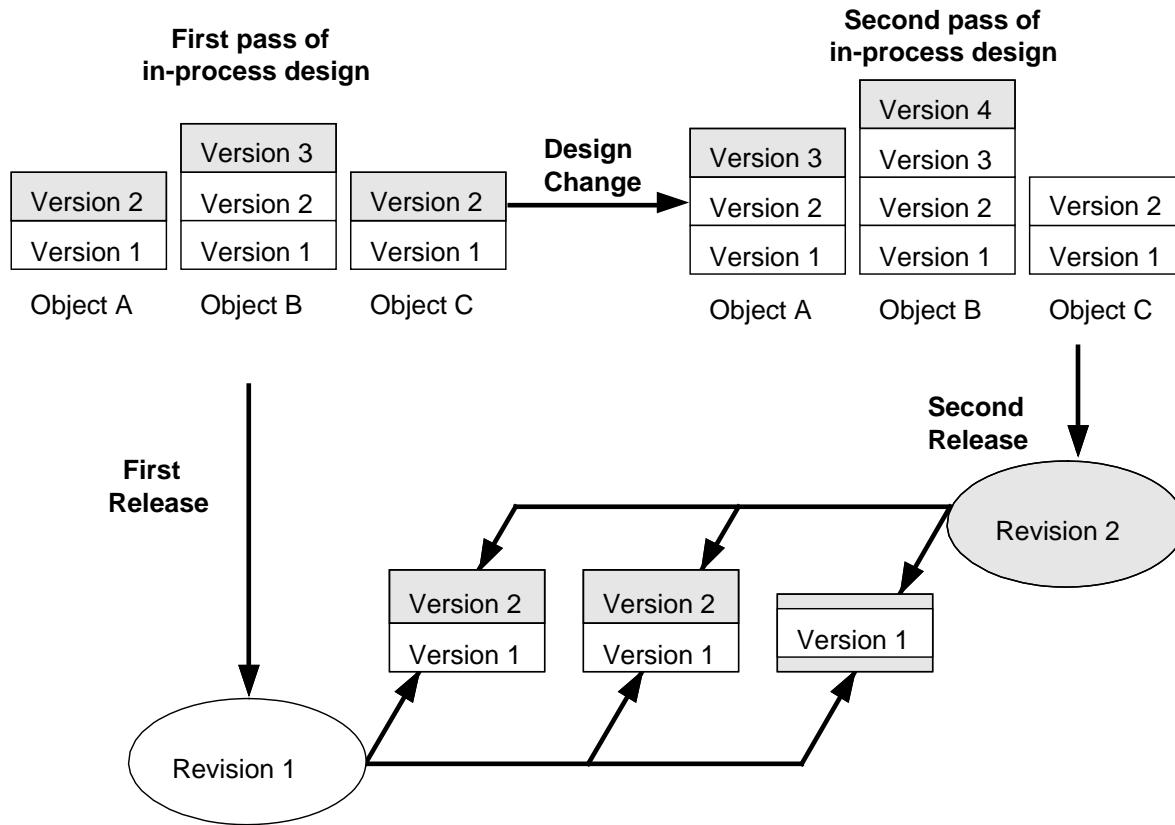
The copied configuration represents Revision 2 of the design, and unlike the released design, you can open and edit the design objects in the copy.

Examples

Figure 2-19 shows an example of creating multiple revisions of a design by releasing a configuration multiple times. In this example, the design consists of objects A, B, and C. Before the first release, assume that you have created several versions of each design object (current versions of objects A, B, and C are 2, 3, and 2, respectively) and that you have created a configuration that contains the current version of each object. When you release the configuration, you attach the annotation “Revision 1” to the release. This annotation is actually

a version property on the released configuration object that defines the configuration. As shown in Figure 2-19, release “Revision 1” consists of versions 1, 1, and 1 of objects A, B, and C, respectively.

Figure 2-19. Example of a Release of Multiple Revisions



Next, assume that you modify the design and, as a result, create version 3 and 4 of objects A and B, respectively. Because C has not changed, version 2 is still its current version. You release the design again, this time attaching the annotation “Revision 2” to the release. This release adds version 2 to released object A and version 2 to released object B. Because object C has not changed since the last release, its version number is not incremented.

Related Topics

[Change Your Session Setup](#)

[Managing Designs](#)

[\\$release_configuration\(\)](#)

[Getting Information about a Release](#)

Freezing a Configuration

You can freeze or unfreeze all design object versions that are represented in the configuration.

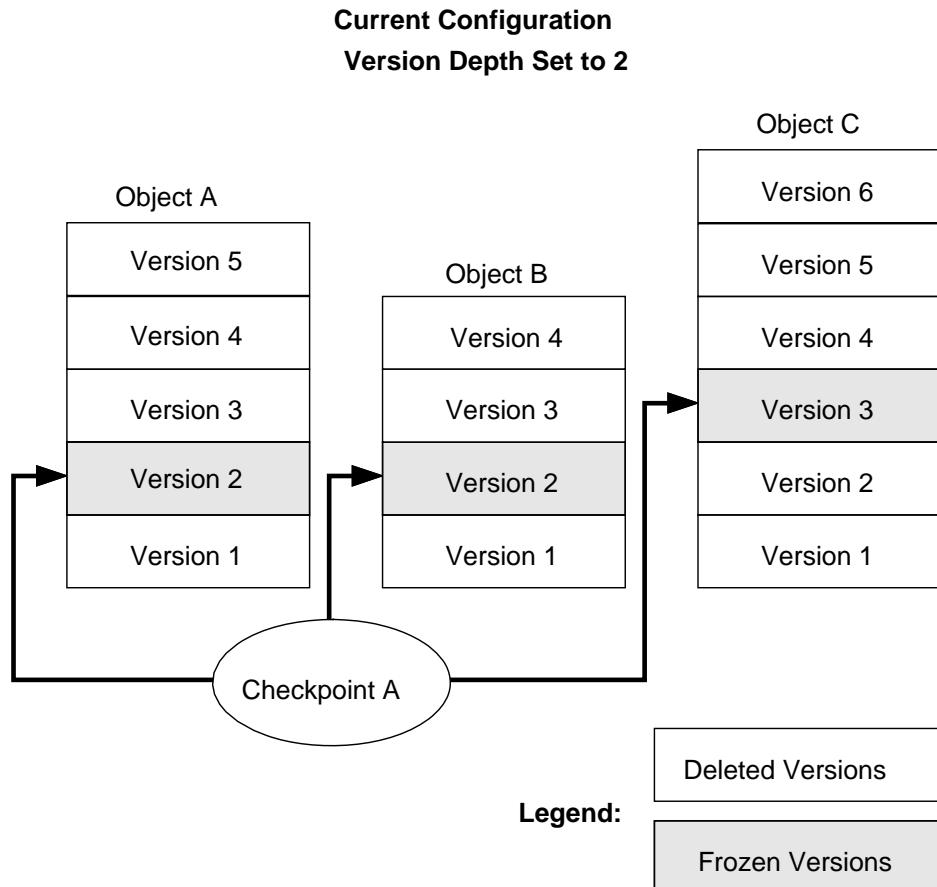
Freezing the configuration freezes each design object version (each entry) in the configuration and creates and saves a new version of the configuration object. Freezing a version prevents it from being deleted (or pruned) by the version depth mechanism as the design objects evolve.

Freezing a design object version does not prevent that version from being deleted with the Delete Object or Delete Configuration command. It only prevents deletion by the version depth mechanism. Unfreezing a configuration allows the version depth mechanism to prune the versions contained in the configuration the next time that someone opens and evolves the corresponding design object.

You can use the Freeze Configuration command to create a “checkpoint” of a design. A checkpoint is simply a snapshot of the current version of the design, protected through version freezing. Freezing a configuration differs from a release, which copies the design to another directory, layers it with past releases, and prevents you from evolving the released objects.

Figure 2-20 shows an example of a checkpoint.

Figure 2-20. Checkpoint of an Evolving Design



In this example, objects *A*, *B*, and *C* compose a configuration that has been frozen at some point in the past. Note that you can still evolve the design. If you unfreeze the configuration, the checkpoint is deleted as *A*, *B*, and *C* evolve.

Related Topics

[Freezing and Unfreezing a Design Configuration](#)

Deleting a Configuration

In a single operation, you can delete every version of a design object that is represented in the configuration.

Because each entry is a single version of a design object, deleting the configuration deletes only the version of the design object that is contained in the configuration. However, if the configuration entry is the design object's only version, or if the entry is an unversioned object, the Delete Configuration command deletes the entire design object. In addition to deleting the configuration entry, the Delete Configuration command also deletes all of the design objects referenced by the configuration entry.

Related Topics

[Configuration Operations](#)

Changing the References of a Configuration

In one operation, you can change the pathnames of the references that are held by all entries in the configuration.

The Change Configuration References command works the same as the Change Object References command. You specify a pattern to be searched for and a pattern with which to replace it. This command is useful when you must update the references of a related set of objects. A typical case where you would want to update the references of a related set of objects is when a design object, that a set of objects references, is moved.

The Change Configuration References command allows you to specify the searched-for pattern with UNIX System V wild cards.

The Change Configuration References command can be executed with a @preview switch, which reports the expected results of an execution without actually executing the command. This command reports and displays real-time status in the monitor window.

If your session setup values specify to show the monitor window, the monitor window is opened when you execute this function. The monitor window remains visible until you return to the

configuration window, using the monitor window's popup menu item **Hide Monitor**. You can revisit the monitor window using the configuration popup menu item **Show Monitor**.

Related Topics

Change Your Session Setup

Regular Expressions

Changing the References of a Design Configuration

Locking a Configuration

In a single operation, you can lock the entire configuration.

Locking the configuration prevents others from editing any of the entries. You typically lock a configuration prior to a lengthy configuration operation such as copy or release. For these operations, the integrity of the data is critical. So, to prevent others from changing any configuration entry while you are copying or releasing, you lock the configuration prior to executing these operations. When you no longer need the configuration locked, you should unlock it.

Related Topics

Lock a Design Configuration

Integrated Design Management (iDM)

You can use the Integrated Design Management (iDM) functionality to manage the design data within Mentor Graphics applications in addition to Pyxis Project Manager.

iDM helps you manage your designs by providing you with an easy way to copy, move, delete, and change references on your designs from within applications such as Pyxis Schematic and Pyxis Custom Analyzer. iDM also provides you with the ability to navigate through your designs and to view and interact with your data's design hierarchy.

iDM manages your design data in the same way that the Pyxis Project Manager manages your design data. Before reading the following text, you should be familiar with the Pyxis Project Manager design management concepts presented in the section, “[A Design Object](#)” on page 31.

Design Management with iDM

iDM commands provide you with the ability to move, copy, delete, and change the references of your design objects from within your application.

iDM also provides you with the ability to view the design hierarchy of your design data. Each of these iDM commands has an easy-to-use dialog box that includes one or more object browser dialogs.

The object browser dialog provided in each of the iDM dialog boxes allows you to specify your command options using simple “point-and-click” mouse techniques. Every object browser dialog also has a filter option that allows you to simplify your view of your design data. Using the object browser dialog filter option, you can specify that a particular design object or set of design objects is visible or not visible in a object browser dialog list area based on a design object's name and/or its type. You can also specify the order in which design objects are displayed in the object browser dialog's list area.

For more information on dialog navigation, refer to the section “[Navigation](#)” on page 55 in this chapter.

The iDM copy and move commands support interrupt handling. You can interrupt the execution of an iDM copy or move command at any time. When you do so, iDM attempts to reverse the effects of the interrupted operation up to the point of the interrupt. However, if you interrupt an iDM move command while references are being updated, a complete reversal is not possible. In this case, the interrupt is canceled to prevent data corruption.

iDM Copy Operation	132
iDM Move Operation	134
iDM Delete Operation.	137
iDM Change References Operation	137
iDM Hierarchy Window	138
iDM Component Window.	139

iDM Copy Operation

You can copy a design object or a set of design objects to another location from within your EDA application using the iDM Copy Design Object command.

This copy operation leaves the source design objects intact and creates duplicates in the specified destination location.

In a single copy operation, iDM copies each member of the source design object's fileset and all of the design objects contained by that source object to the destination directory. For example, if you copy the component *card_reader*, which contains the component *add_detector*, and *add_detector* contains the schematic *schem*, all three design objects are copied to the destination directory. In the destination directory the design object *card_reader* contains a copy of *add_detector*, and the copy of *add_detector* contains a copy of *schem*.

The iDM copy operation has the following features:

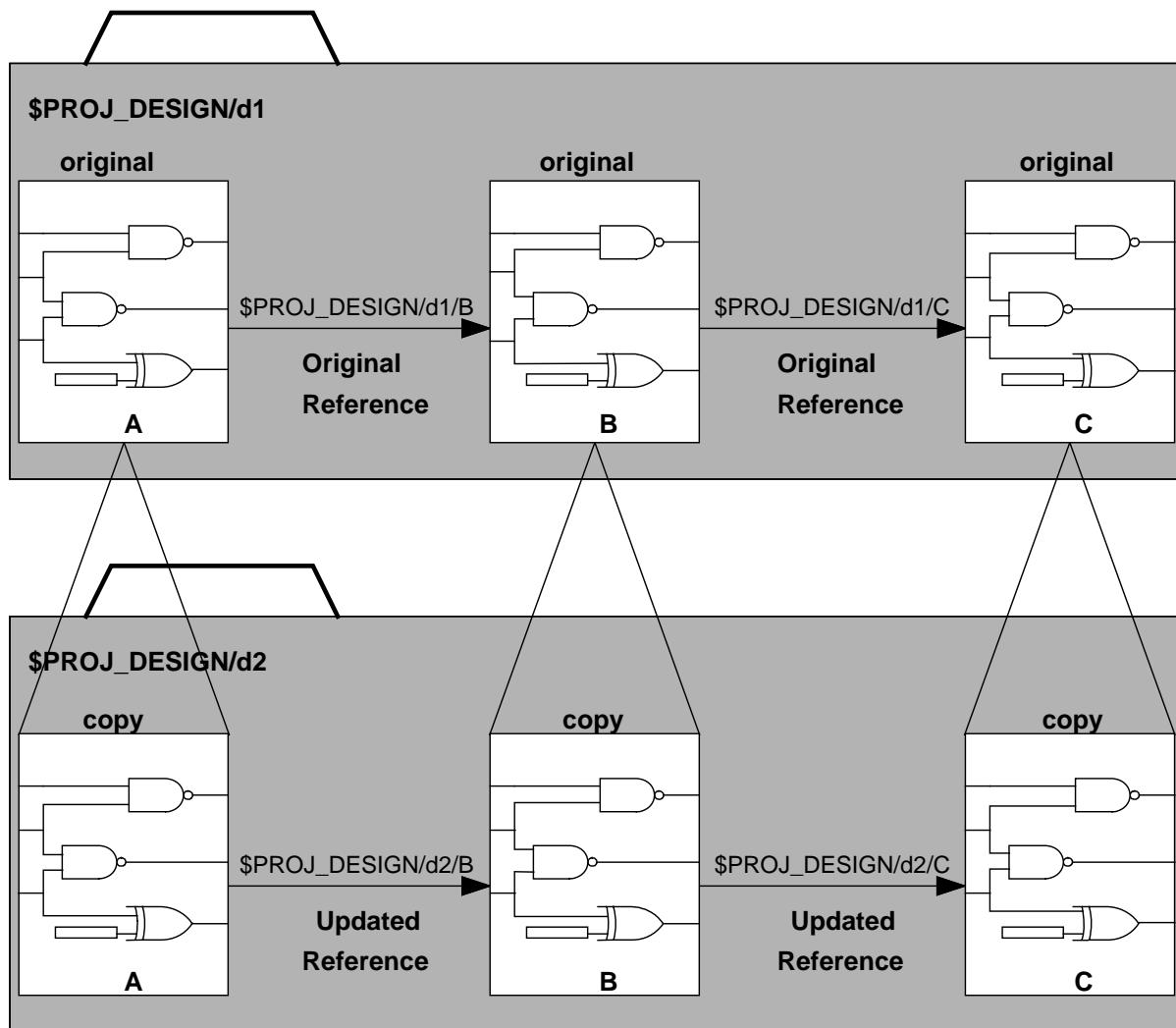
- iDM copies all of the versions of a versioned source design object.

- iDM allows you to specify whether to update the references of the copied design objects to their new location or to leave the references pointing to their original target objects. If you choose to update references during the copy operation, the reference pathnames relative to the root of the design are updated to the new root location.
- If conflicting design objects exist in the destination container, iDM allows you to specify whether to cancel the copy operation or to overwrite the conflicting design objects.
- If the destination container does not exist, iDM allows you to specify whether to cancel the copy operation or to create a destination container at the specified location.
- iDM allows you to interrupt the copy operation, by pressing the Ctrl and Backslash keys together.

Figure 2-21 shows reference updating during a simple iDM copy operation. In this example, a set of design objects, A, B, and C, is copied from directory `$PROJ_DESIGN/d1` to directory `$PROJ_DESIGN/d2`. Note that in directory `$PROJ_DESIGN/d2`, the references of design objects A, B, and C now point to locations in directory d2. When you specify to update references, iDM updates references for you automatically during copy operations. If you had specified not to update references, the references of A, B, and C would still point to locations in directory d1.

When you specify to update references during a copy operation, iDM updates the references *between* the design objects in the selection set. However, iDM does not update references that point to design objects outside the selection set. If you select and copy A in one operation, B in a second operation, and C in a third operation, A, B, and C are not a single selected set, and no reference updating occurs between them. Even if you had copied A, B, and C in three separate copy operations and had specified to update references, `$PROJ_DESIGN/d2/A` would still point to `$PROJ_DESIGN/d1/B`, and `$PROJ_DESIGN/d2/B` would still point to `$PROJ_DESIGN/d1/C` because in each of the three copy operations the references point outside of the selection set.

Figure 2-21. Reference Updating During a Copy



Related Topics

[Versions](#)

[Copying a Design Object](#)

[References](#)

iDM Move Operation

You can move a design object or a set of design objects to another location using the iDM Move Design Object command.

This move operation places the source design objects intact in the specified destination location. When you move a design object, all of the objects contained within the selected design object are also moved.

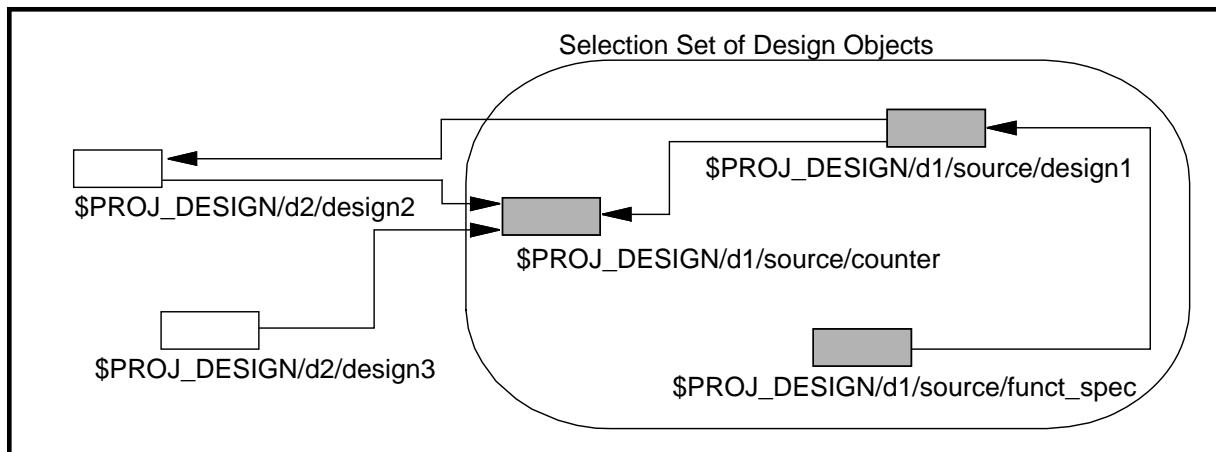
The iDM move operation has the same features as the copy operation, as shown in the following list:

- iDM moves all of the versions of a versioned source design object.
- iDM allows you to specify whether to update the references of the moved design objects to their new location or to leave the references pointing to their original target objects. If you choose to update references during the move operation, the reference pathnames relative to the root of the design are updated to the new root location.
- If conflicting design objects exist in the destination container, iDM allows you to specify whether to cancel the move operation or to overwrite the conflicting design objects.
- If the destination container does not exist, iDM allows you to specify whether to cancel the move operation or to create a destination container at the specified location.
- iDM allows you to interrupt the move operation, by pressing the Ctrl and Backslash keys together.

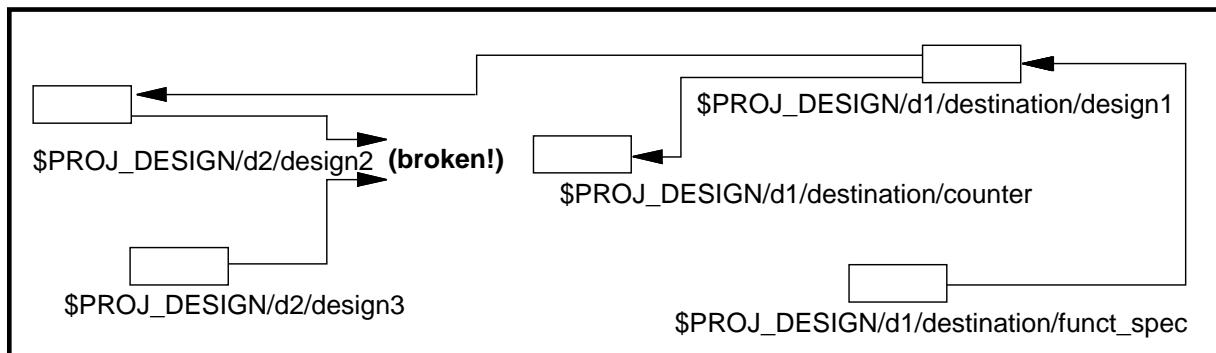
Although the iDM move operation updates the references between design objects in the selected set if you specify to do so, it does not update the external references of other design objects that point to the design objects that are being moved. These external references are broken as a result of moving a design object or a set of design objects to which they point. To use these references after the move operation, you must manually update them with the iDM Change Design Object References command.

Figure 2-22 shows an example of the move operation in which reference updating is specified. In this example, the design objects *design1*, *counter*, and *funct_spec* are moved from *\$PROJ_DESIGN/d1/source* to *\$PROJ_DESIGN/d1/destination*. As you can see, the design objects *design2* and *design3* located at *\$PROJ_DESIGN/d2* are not in the selection set. Because the references from *\$PROJ_DESIGN/d2/design2* and *\$PROJ_DESIGN/d2/design3* still point to *counter* in *\$PROJ_DESIGN/d1/source*, these references are broken.

Figure 2-22. Reference Updating and Breaking After a Move Command



Before Moving the Selected Set



After Moving the Selected Set

Your ability to successfully interrupt the move operation is dependent upon what task is actually being performed at the time you interrupt. If you interrupt the move operation while the data is being moved from the source to the destination, iDM attempts to reverse the actions of the move operation and return the data to its original state. However, if you interrupt the move operation later in the execution when the references of the moved design objects are being updated to their new location, iDM displays an error message stating that you cannot interrupt at this point in the operation and cancels the interrupt. iDM does this to prevent corruption of the design objects' references.

Related Topics

[Versions](#)

[Moving a Design Object](#)

[References](#)

iDM Delete Operation

You can delete a design object or a set of design objects in a single operation using the iDM Delete Design Object command.

The deletee operation removes all of the versions, including any frozen versions, of the selected design objects. If a design object is a container object, deleting the container also deletes all of the design objects that it contains.

The delete operation allows you to specify whether container design objects that contain locked objects should be deleted. This feature safeguards you against mistakenly deleting an important design object that exists inside a container object. If you specify not to delete containers that contain locked objects, all of the specified design objects in the selection set, *except* for the container object containing the locked object, are deleted.

Deleting a design object using the Delete Design Object command is a permanent deletion. You cannot retrieve a design object that you have deleted with this command.

The iDM delete operation has the following features:

- iDM deletes all of the versions of a versioned source design object.
- iDM allows you to specify whether container design objects that contain locked objects are deleted.
- iDM deletes the specified design objects permanently and they cannot be retrieved.

Related Topics

[Versions](#)

[Salvaging a Design Object](#)

[References](#)

[Read-only and Write Locking Modes](#)

[Delete a Design Object](#)

iDM Change References Operation

You can change the references of a design object or a set of design objects using the iDM Change Design Object References command.

The change references command updates the pathnames of all of the references that are held by all of the versions of the specified design objects and by all of the design objects that they contain.

The iDM Change Design Object References command allows you to use System V regular expressions to specify multiple pathname strings to match and to replace. In each reference, the change references operation attempts to match each of the specified pathname strings until a successful match is found. When a successful match is found, the change references command replaces the matched pattern with the replacement string and then proceeds to the next

reference. If a reference contains more than one matching string, only the first match is changed.

The iDM Change Design Object References command function changes the references of design objects, as they are actually stored in the reference. Additionally, if you specify a hard pathname replacement pattern, the specified hard pathname is not converted to a soft pathname before it is stored in the reference.

The iDM Change Design Object References command allows you to specify whether you want to lock your source design objects before changing their references. If you specify that your source design objects should not be locked, the change references operation executes faster. However, you should only use this option when you are absolutely sure that the source design objects will not be changed by another user during the operation's execution.

Caution



If the source design objects are modified during the execution of the Change Design Object References operation, they can be corrupted.

The iDM change references operation has the following features:

- iDM updates the pathnames of the references that are held by all of the versions of the specified design objects and by all of the design objects they contain.
- iDM only replaces the first successful match in each reference.
- iDM allows you to use System V regular expressions to specify your source and replacement strings.
- iDM does not convert hard pathname replacement strings to their soft pathname equivalent before storing them in the references.
- iDM allows you to specify whether your source design objects are locked before the change references operation is executed.

Related Topics

[Versions](#)

[Pathnames in the Pyxis Project Manager](#)

[References](#)

iDM Hierarchy Window

Integrated Design Management's Hierarchy Window capability allows you to view your design's configuration and component information without having to exit your current application.

By doing so, you can also:

- Display a component hierarchy in the context of a specific viewpoint
- Display hierarchies that are not dependent on viewpoints
- Dynamically update the list window, giving you the current list of levels rather than having to open a new window to display the updates
- Display instance information for a given component
- Display the path to a single occurrence
- Display the hierarchy as an indented list or a graphical tree
- Probe other applications.
- Display the value of a specified property rather than the instance name next to component in the hierarchy listing.

The Hierarchy displayed is a physical or logical hierarchy as it is defined by the component level of the design. This allows you to view directly information concerning the instance names, property values, object designations and model information.

Related Topics

[Versions](#)

[References](#)

iDM Component Window

The Component Window allows you to view and/or edit information about specific components in your design.

Editing capability is only available from the following applications: Pyxis Project Manager, Pyxis Schematic, DVE, and AutoLogic. Regardless of whether you need to edit or just view the component structures, the Component window enables you to do the following:

- List the Part Interfaces and displays its Models, Pins and Body Properties for a given component.
- Register and unregister component models.
- Add/delete or edit labels for a component.
- Show all objects contained by the component, and filter out objects depending on type.

Related Topics

[Versions](#)

[References](#)

Chapter 3

Operating Procedures

Pyxis Project Manager uses a graphical interface to help you perform design management tasks.

Open and Close the Pyxis Project Manager	141
Pyxis Project Manager's Graphical Interface	142
Design Navigation	157
Tool Invocation	162
Design Objects	169
Versions	197
References	205
Design Object Properties	222
Change Your Session Setup	224
Managing Designs	238
Managing Designs Using iDM	279
Design Management in the Operating System Shell	302
Customizing Your Design Management Environment	306
Installing Process Design Kits (PDKs)	307
ICanalyst Projects	308

Open and Close the Pyxis Project Manager

Use the command line to invoke the tool.

Open the Pyxis Project Manager

To invoke the Pyxis Project Manager, enter the following command at the shell window prompt:

```
$MGC_HOME/bin/dmgr_ic
```

The Pyxis Project Manager session window should appear. If it does not, contact your system administrator.

i **Tip:** For details on the **dmgr** shell command and the options that you can use with it, refer to the Shell Command Dictionary in the *[Pyxis Project Manager Reference Manual](#)*.

Close the Pyxis Project Manager

To close the Pyxis Project Manager, choose the **MGC > Exit** pulldown menu option.

Related Topics

[Startup Files](#)

Pyxis Project Manager's Graphical Interface

Use the toolbar and menus in the GUI to perform common tasks.

Mouse Functions	142
Session Window	146
Updating a Window	152
Setup Pulldown Menu Options	153

Mouse Functions

Use the mouse to select and manipulate items in the Pyxis Project Manager windows.

- Item Selection
 - To select items in iconic windows, position the mouse pointer anywhere on an item's icon, and click the Select (left) mouse button.
When you select an object in an iconic window using the Select mouse button, all previously selected objects are unselected.
 - To select an item in a list window, position the mouse pointer anywhere on the item's icon or name, and click the Select (left) mouse button.
When you select an object in a list window using the Select mouse button, all previously selected objects are unselected.
- Item Dragging
 - You can use dragging to rearrange icons in an iconic navigator, change the toolbox search path, or move design objects from one navigator to another. Similarly, you can copy design objects from one navigator to another by dragging them while depressing the Shift key. Dragging an item means selecting it and moving the mouse pointer to a new location, while keeping the Select mouse button depressed. When you do this, the selected item tracks the mouse pointer.
- Multiple Item Selection
 - In every Pyxis Project Manager window, you can select several items simultaneously. The four methods of multiple selection are:

- Drawing an *extent box*
You use extent boxes, which are dynamic rectangles created using the mouse, in iconic navigator, tools, and trash windows.
- Sweeping a list area
You use list sweeping in list windows, including the list navigator and the configuration window.
- Clicking the Select (left) mouse button, while pressing the Ctrl key
You can click the Select mouse button while pressing the Ctrl key over a design object in any window.
- Executing the Select Object command
You can use wild cards to specify your selection set in any window.

Selecting a Rectangular Region	143
Sweeping a List	144
Selecting and Unselecting Multiple Items	144
Selecting by Command	144

Selecting a Rectangular Region

Draw an extent box.

Procedure

1. Move the mouse pointer to an area of the iconic window that is *not occupied* by an icon or a name.
If you attempt to start an extent box with the mouse pointer over an icon or name, you do not get an extent box. Either nothing happens, or you can drag the icon on which the mouse pointer rests.
2. Press the Select mouse button, and keep it depressed.
3. While keeping the Select mouse button depressed, move the mouse pointer so that the items that you want to select are inside of the box.
4. A box rubber-bands with the moving mouse pointer.
5. Release the Select mouse button.

The items inside the box are selected.

Related Topics

[Mouse Functions](#)

Sweeping a List

Select all objects touched by the mouse pointer.

Procedure

1. To sweep a list, keep the Select mouse button depressed.
2. Move the mouse pointer slowly over the list, either down or up.

Related Topics

[Mouse Functions](#)

Selecting and Unselecting Multiple Items

Select and unselect multiple items one at a time.

Selecting items with Ctrl-click allows you to select multiple items one at a time, with each selection a separate operation. As a result, you can select multiple items that are not near each other.

Procedure

1. Select the first object by clicking it.
2. Position the pointer on the next object, press the Ctrl key and click on that object, for each additional selection.
3. After you select multiple items, you can unselect individual items by pressing the Ctrl key and clicking the item's icon, or, in a list area, the item's icon or name.

Related Topics

[Mouse Functions](#)

Selecting by Command

Execute the Select Object command.

Procedure

1. Press the space bar to display the popup command line and enter **SElect OBject**.
2. Press the Return key to display the Select Object prompt bar and enter a text string, which matches the names of the objects you want to select, in the Name file.

To select multiple objects, you must specify the name using UNIX System V wild cards.

3. Enter the type of the objects you wish to select, in the Type field.

Entering a value in the Type field is optional. However, you can use this value as a means of filtering out objects of undesired types.

4. Select the selection mode you want, by clicking on the arrows.

By default, the Select Object command unselects any previously selected objects and selects the newly specified objects. If you want to add the objects you are currently selecting to a previous selection set, specify the Add mode.

5. Click the **OK** button.

All design objects whose name matches the text string you specified in the Name field, and whose type matches the type specified in the Type field are selected.

You can unselect all objects by choosing **Unselect All** from the navigator's popup menu, or by pressing the Escape key.

Related Topics

[Mouse Functions](#)

[Regular Expressions](#)

Accessing Popup Menus

In the Pyxis Project Manager every window has its own popup menu which provides commands specific to that window.

The three methods for accessing popup menu commands are:

- Displaying and executing the popup menu command

To display the active window's popup menu, position the mouse pointer anywhere inside the window, and press the Menu (right) mouse button.

- Re-displaying the last popup menu you displayed

To re-display the last popup menu you displayed in the active window, position the mouse pointer anywhere inside the active window and, while holding down the Shift key, press the Menu (right) mouse button.

The last popup menu that you displayed is re-displayed in the window with the last command you executed highlighted.

- Executing the last popup menu command you executed

To repeat the last popup menu command that you executed, position the mouse pointer anywhere inside the window and, while holding down the Ctrl key, press the Menu (right) mouse button.

The last popup menu command that you executed by using a popup menu item is executed again.

Related Topics

[Mouse Functions](#)

Session Window

The Pyxis Project Manager uses windows, popup menus, and toolbars to control functionality.

Pulldown and Popup Menus	146
Toolbar Menus	147
The Session Window Display	149
Windows Pulldown Menu Options	152

Pulldown and Popup Menus

You interact with the Pyxis Project Manager through windows and menus on your screen.

Examples of Pyxis Project Manager windows include the session window, the tools window, and the trash window.

Every window has its own pulldown menus, which the Pyxis Project Manager displays in the menu bar. As you activate different windows, the menu bar changes to show the available commands for the active window.

You can access a pulldown menu by using one of the following two methods:

- To access a pulldown menu using the mouse, position the pointer on a menu name in the menu bar and press the Select (left) mouse button.

A menu appears, displaying all the menu items available for the menu name you selected. Many of the menu's items, themselves, have menus; these are referred to as cascading menus. The presence of a cascading menu is symbolized by a “>” following the menu item's name. By moving the mouse cursor over the menu item you wish to select, you may access either the menu item or its cascading menu.
- To access a pulldown menu using a function key, press the F10 function key and type the underlined letter which corresponds to the menu option you want to select. You can access cascading menus by continuing to enter the underlined letter which corresponds to the menu option you want.

Every window also has its own popup menu, which the Pyxis Project Manager displays at the current mouse pointer position. The popup menu that you see depends on which window you are in when you pop it up. You can access a popup menu by using one of the following two methods:

- To access a popup menu using a mouse button, position the pointer in the window whose popup menu you want to access and press the Menu (right) mouse button. A menu appears, displaying all the menu items available for the menu name you selected. Many of the menu's items, themselves have menus; these are referred to as cascading menus. The presence of a cascading menu is symbolized by a “>” following the menu item's name. By moving the mouse cursor over the menu item you wish to select, you may access either the menu item or its cascading menu.
- To access a popup menu using a function key, press the F10 function key and type the underlined letter which corresponds to the menu option you want to select.

You can access cascading menus by continuing to enter the underlined letter which corresponds to the menu option you want.

The Pyxis Project Manager provides the most commonly used menu items in the popup menu, instead of requiring you to move your mouse pointer to the pulldown menu. The pulldown menu of a window always contains all of the commands that are available in the window's popup menu and frequently contains more. In both popup and pulldown menus, unavailable menu items are grayed out.

Related Topics

[Session Window](#)

Toolbar Menus

There are five toolbars in Pyxis Project Manager: **Windows, Interface, Pyxis Project Manager, Configuration Window**, and the **Project Navigator** toolbars.

Each toolbar provides a set of options that are specific for a task, and generally opens up a dialog box or prompt bar upon clicking any of the icons in it. All the toolbar options can also be accessed through the pulldown and/ or popup menus.

To use the toolbars, simply double-click the option. The toolbars can be viewed in three modes: **Icon/Text**, **Icon Only**, and **Text Only**. The toolbar modes can be changed by selecting the mode through the pulldown menu **Setup > Toolbars**. In addition, when you move the mouse over any toolbar icon in any of the viewing modes, a tooltip appears with the icon description.

Windows Toolbar

The Windows toolbar icons execute session related tasks and bring up the various windows.

The Windows toolbar displays the following task icons: Setup Session, Setup Monitor, Session Monitor, Hide Transcript, Open Navigator, Project Navigator, Hierarchy Window, Component Window, Configuration Window, and Trash Window. [Figure 3-1](#) illustrates the Windows toolbar in Icon Only mode. [Figure 3-2](#) illustrates the Windows toolbar in Icon/Text mode.

Figure 3-1. Windows Toolbar (Icon Only mode)**Figure 3-2. Windows Toolbar (Icon/Text mode)**

Interface Toolbar

The Interface toolbar is part of the user interface that is common to all Pyxis Custom Design flow applications.

The Interface toolbar displays the following task icons: Load Profile, Save Profile, Save Profile As, Remove Profile, Restore Default Profile, and Register Commands. [Figure 3-3](#) illustrates the Interface toolbar in Icon Only mode.

Figure 3-3. Interface Toolbar (Icon Only mode)

Pyxis Project Manager Toolbar

The Pyxis Project Manager toolbar icons execute Pyxis Project Manager-specific tasks and operate on design objects.

The Pyxis Project Manager toolbar displays the following task icons: Refresh All, Report Info, Release Design, Copy Object, Move Object, Delete Object, Check References, and Rename Object. [Figure 3-4](#) illustrates the Interface toolbar in Icon Only mode.

Figure 3-4. Pyxis Project Manager Toolbar (Icon Only mode)

Configuration Window Toolbar

The Configuration Window toolbar icons are enabled only when a Configuration Window is opened, and operate on configuration entries.

The Configuration Window toolbar displays the following task icons: Add Entry, Remove Entry, Retarget, Build Rules, Report Info, Build, Release Configuration, and Copy Configuration. [Figure 3-5](#) illustrates the Configuration Window toolbar in Icon Only mode.

Figure 3-5. Configuration Window Toolbar (Icon Only mode)



Project Navigator Toolbar

The Project Navigator toolbar icons execute project-specific tasks and operate on design objects within the hierarchy of a project.

The Project Navigator toolbar displays the following task icons: Open (Hierarchy), Project, External Library, Technology Library, Library, Category, Technology Category, Cell, Layout, Symbol, and Schematic. [Figure 3-6](#) illustrates the Project Navigator toolbar in Icon Only mode.

Figure 3-6. Project Navigator Toolbar (Icon Only mode)



Related Topics

[Session Window](#)

[Project Navigator Popup Menu](#)

The Session Window Display

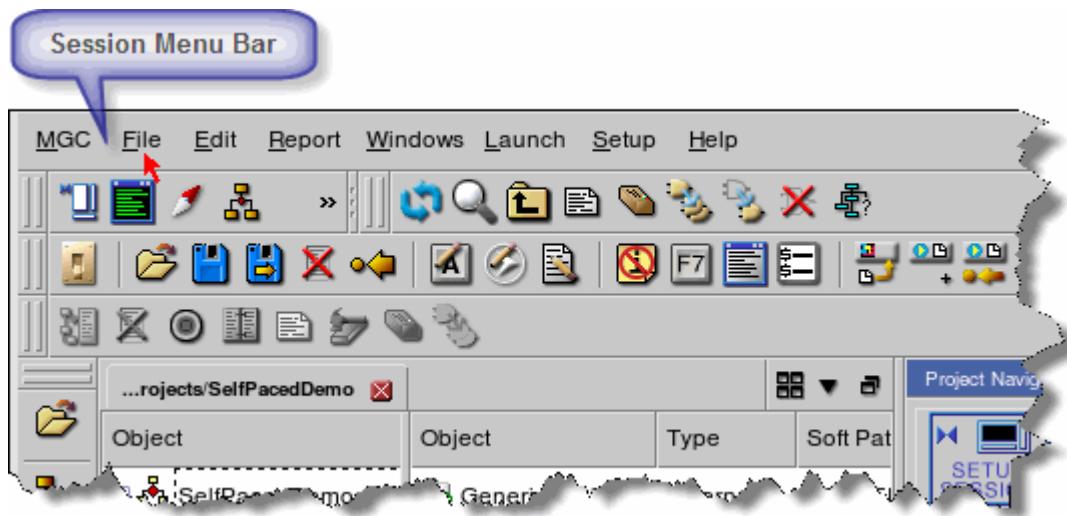
The Pyxis Project Manager session window is the first window that appears when you invoke the Pyxis Project Manager.

When this window first appears, you usually want to perform one of the following actions:

- Customize your Pyxis Project Manager session
- Open a navigator
- Open a tools window
- Open a configuration window to begin your design work

You can also set the Pyxis Project Manager to automatically open a navigator and/or a tools window, using startup files.

Figure 3-7. Pyxis Project Manager Session Window



Session Window Menu Bar

The session window pulldown menu bar, shown in Figure 3-7, is available inside an active session window. Six pulldown menus are available from the session menu bar: **MGC**, **File**, **Edit**, **Setup**, **Windows**, **Launch**, and **Help**. Each of the other Pyxis Project Manager windows have different pulldown menus. However, some of the session pulldown menus are available from other windows.

The **Setup** menu, which is one of the session pulldown menus that is available in all the Pyxis Project Manager windows, provides options for customizing your Pyxis Project Manager session.

The **File**, **Edit**, and **Windows** menus provide options for opening Pyxis Project Manager windows and managing your design data. Many of the options in these three pulldown menus are also available in window popup menus.

The **MGC** and **Help** menus, are common to the windows of all Mentor Graphics' Pyxis Custom Design flow applications. The **MGC** menu provides options for opening notepads, redrawing windows, managing userware, displaying a transcript window, setting session defaults, accessing the iDM functions, and accessing the location map management functions, as well as the provision to export the screen to an output file.

Although the **Help** menu is common to all Mentor Graphics' Pyxis Custom Design Flow applications, the Pyxis Project Manager provides additional menu items on the **Help** menu to assist you in learning about Pyxis Project Manager. The menu path to these additional menu items is **Help > Open User's Manual| Open Reference Manual| Open PDF Bookcase | Search| Support| About Pyxis Project Manager**.

Session Window Popup and Pulldown Menus

The session window popup menu provides the following menu, which can be accessed under the **Windows** pulldown menu as well:

- Open Navigator > View by Icon | View by Name
This menu item opens a navigator in iconic or list viewing mode.
- Open Tools Window
This menu item opens a tools window. The tools window displays, as icons, all of the tools that you can invoke from the Pyxis Project Manager session.
- Open Project Navigator
This menu item opens up a project navigator window. This two-paned explorer/ view window model displays hierarchy on the left in the explorer pane and the contents of the current hierarchical location on the right in the view pane. Both these areas provide a context menu which allows easy access to the common commands for operating on the selected objects.
- Open Configuration > New | Existing
This menu item opens an untitled configuration or brings up a dialog box in which you specify the pathname of an existing configuration object. After you execute the **OK** button, a configuration window appears. You use the configuration window to create or to modify a configuration object.
- Open Trash Window
This menu item opens a trash window. This window displays design objects that you have dragged to the trash can. You can view objects in the trash can before deleting them. You can recover an object from the trash window, by using the trash window's popup menu item **Untrash Object**. After you "empty the trash," this window becomes empty, indicating that your deletion was successful.
- MGC > Notepad | Cleanup Windows | Userware | Transcript | Setup | Design Management | Location Map | Export Screen
The MGC menu item provides menu options available to all Mentor Graphics applications.

Related Topics

[Change Your Session Setup](#)

[\\$\\$set_working_directory\(\)](#)

[\\$change_location_map_entry\(\)](#)

[Design Navigation](#)

[Design Management with Location Maps](#)

[\\$\\$show_location_map\(\)](#)

[\\$read_map\(\)](#)

[Tool Invocation](#)

Windows Pulldown Menu Options

Apart from the popup menu options explained in the previous section, there are the following additional pulldown menu items available in the **Windows** pulldown menu.

These are specific to the Project Navigator window.

- Open Types Window

To create a type, you need to have administrator permissions. However, you can and filter out which of the existing types you want to include or not, depending on the properties that the type holds.

Clicking on the **Windows > Open Types Window** option simply brings up the “Type Manager: Known Type Reps” window. This window displays a sorted list of all of the design object types known to Pyxis Project Manager, with no duplicates.

- Open Session Monitor

The Session Monitor contains the transcripts of the session history, including any errors and validations for any of the tasks in the Project Navigator Window.

- Open Hierarchy Window

This opens up a dialog box prompt that allows you to type in or browse to an existing project hierarchy in Project Navigator.

- Open Component Window

This contains information about the component hierarchy, which is displayed in a physical or logical hierarchical format, as it is defined by the component level of the design. The Hierarchy Window displays this information graphically so that you can view directly, information concerning the instance names, property values, object designations and model information.

Related Topics

[Updating a Window](#)

Updating a Window

Updating a window provides you with the most recent display of the window's contents.

For example, if you have two navigators running side by side and displaying the same location, and you delete a design object from one of them, the other navigator still displays the deleted design object. You need to update the navigator window to remove the deleted object from its display.

The iconic navigator calculates the number of columns based on the window width, instead of using a “fixed” number of columns. As a result all icons are displayed in the visible window column and you can view any icon by scrolling up and down through the window.

The iconic navigator recalculates the number of columns needed to properly display the total number of icons in the navigator's current directory every time you navigate to a new directory or execute the **Update Window** command. Simply redrawing or resizing the window does not cause the number of columns to be recalculated.

Procedure

1. Activate the navigator from which the design object was not deleted.
2. Select the **Update Window** menu option from the window popup menu, or the **View > Update Window** menu item from the pulldown menu.

The deletion is reflected in the active navigator. After this updating procedure, both navigators display the same contents.

Related Topics

[Pyxis Project Manager's Graphical Interface](#)

Setup Pulldown Menu Options

The Pyxis Project Manager menu bar provides the following menu options under the **Setup** pulldown menu item:

- **Admin Login**

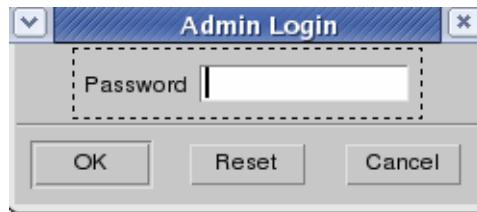
The Admin Login is used to enable custom types and tools to be created and edited inside of the directory specified by the \$MGC_CUSTOM_TYPE_DIR system environment variable. In order to login as an administrator, perform the following steps:

- a. Before launching Pyxis Project Manager, set the \$MGC_CUSTOM_TYPE_DIR environment variable to a stable location in your system. For example, you can set it thus:

```
setenv MGC_CUSTOM_TYPE_DIR /site/dmgr_admin_dir
```

- b. In your Pyxis Project Manager session, login as the administrator using the pulldown menu function **Setup > Admin Login**.
- c. This brings up the Admin Login dialog box, as shown in [Figure 3-8](#).

Figure 3-8. Admin Login Dialog Box



d. The default password is "admin1234".

- **Admin Logout**

Use the Setup > Admin Logout menu item to log out of administration model.

- **Check Registries**

The **Setup > Check Registries** option brings up the Check Registries dialog box.

- **New Type**

You can only create custom types in administrator mode. To create a new type, use the pulldown menu option **Setup > New Type**. The New Type prompt bar appears, as shown in [Figure 3-9](#).

Figure 3-9. New Type Prompt Bar



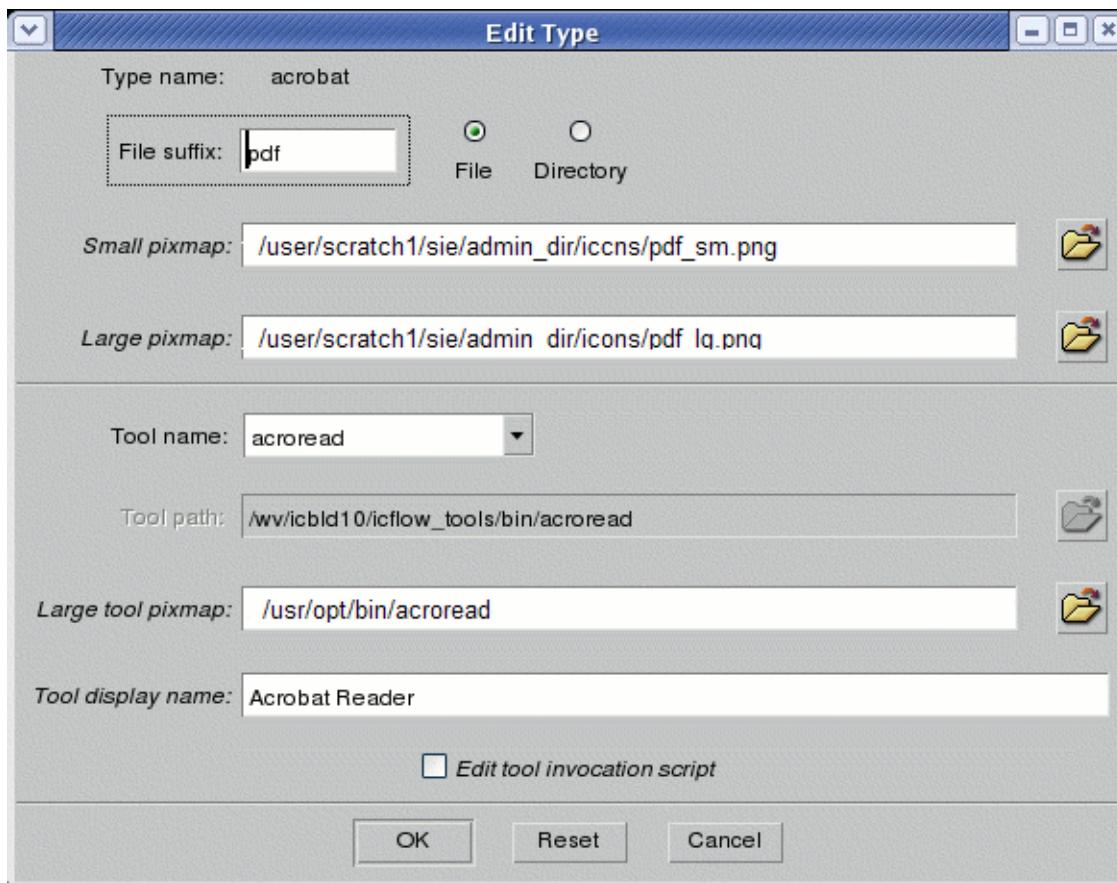
Enter the name of the new type at the prompt, and click OK to execute the prompt bar.

- **Edit Type**

You can only edit custom types in administrator mode.

To edit a new type, access the pulldown menu option **Setup > Edit Type**, which brings up the Edit Type dialog box, as shown in [Figure 3-10](#).

Figure 3-10. Edit Type Dialog Box



The options have the following meaning:

- a. File suffix: This defines how the type is recognized.
- b. File/ Directory: This specifies whether the object is a file or a directory.
- c. Small pixmap (optional): This is the small icon that is used to identify the design object in the Project Navigator window, session browser, and the Navigator window, when the "view by name" option is chosen.
- d. Large pixmap (optional): This is the large icon that is used to identify design objects in the Navigator window, when the "view by icon" option is chosen.
- e. Tool name (optional): This is the name of the tool that is used to open the type.
- f. Tool path: This is the system toolpath that is used to invoke the tool.
- g. Edit tool invocation script (optional): Choosing this option brings up the Tool Qualification Script for Edit directory, after you exit the Edit Type dialog box.
- h. Click OK to execute the Edit Type dialog box.

Note



For more information on type registries and adding a toolbox with custom tools, refer to Chapter 2.

- **Edit Tool Type**

You can only edit tool types when you are in administrator mode.

To edit a new type, access the pulldown menu option **Setup > Edit Tool Type**.

- **Preferences**

The **Setup > Preferences** option is part of the Common User Interface and is therefore common to all Pyxis Custom Design flow applications.

- **Hotkeys, Strokes, Macros, and Registered Commands.**

These options are part of the Common User Interface.

- **Toolbars**

The **Setup > Toolbars** option is part of the Common User Interface and is therefore common to all Pyxis Custom Design flow applications. The toolbars can be viewed in either iconic, or text or icon/text modes.

By default, the Windows, Pyxis Project Manager, and Project Navigator toolbars are displayed in icon mode.

- **Windows**

The **Setup > Windows** option is also part of the Common User Interface and is therefore common to all Pyxis Custom Design flow applications.

By default, the Message Area is always displayed.

- **Workspace**

The **Setup > Workspace** option is another menu item that is part of the Common User Interface and is therefore common to all Pyxis Custom Design flow applications.

By default, the Use Additional Workspaces and Default Workspace options are selected.

Related Topics

[*\\$login_admin\(\)*](#)

[*Problems with Type Registries*](#)

[*\\$logout_admin\(\)*](#)

[*Pyxis Common User Interface User's Manual*](#)

Design Navigation

Design navigation is one of the most important features of the Pyxis Project Manager, along with customized tool invocation and design data configuration management. When you navigate in the Pyxis Project Manager, you view design objects, which are the basic elements of data for Mentor Graphics designs.

From the Pyxis Project Manager navigators, you can perform many important design management tasks between tool sessions. These tasks include opening, deleting, moving, and adding references to design objects. In fact, most operations that work on a single design object are found in the navigator.

Because the navigator's popup menu contains most of the operations that you perform on design objects, those items are described in the section “[Design Objects](#)” on page 169.

Location Map	157
Open a Navigator	157
Search For a Design Object	159

Location Map

A location map is an ASCII file that contains soft prefixes that represent the main directories of your data, and their associated hard pathnames.

A location map relates soft names that logically represent data repositories to one or more hard pathnames, each of which lead you to the actual location of the data repository. Each project team has a project location map that identifies all their primary data repositories. Additionally, a master location map that represents a central point for supporting interoperability at a work site, or across a reasonably localized computer network, is administered by the master location map administrator.

The Pyxis Project Manager allows you to navigate either with or without a location map. Each of these methods has constraints which govern its use.

Related Topics

[Design Management with Location Maps](#)

Open a Navigator

Every navigator has two viewing modes: iconic mode and list mode.

The iconic mode displays design objects as a matrix of large icons, with each design object's name beneath its icon. The list mode displays design objects as an alphabetically sorted list of small icons, with each design object's name to the right of its icon. After you open a navigator in either viewing mode, you can switch between the two modes.

The following list presents instructions on how to open both an iconic and a list navigator and how to switch between viewing modes:

- To invoke an iconic navigator, execute **Open Navigator > View by Icon** from the session window's popup menu. Alternately, execute **Windows > Open Navigator > View by Icon** from the session window's pulldown menu.

The iconic navigator appears. The title bar displays the pathname of the directory whose contents the navigator currently displays. You can also resize and move the iconic navigator.

- To invoke a list navigator, execute **Open Navigator > View by Name** from the session window's popup menu. Alternately, execute **Windows > Open Navigator > View by Name** from the session window's pulldown menu.

The list navigator appears. The title bar displays the pathname of the directory whose contents the navigator currently displays. You can resize and move the iconic navigator. Figure 3-11 shows a list navigator on the right.

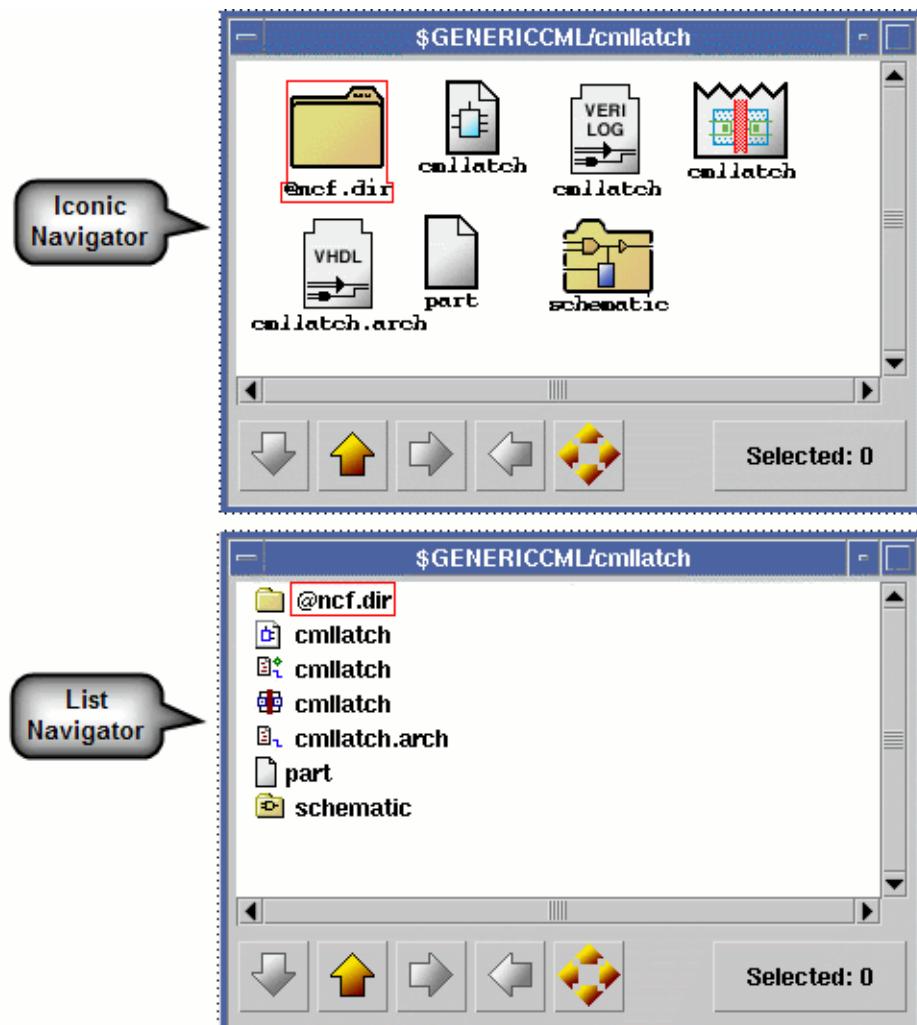
- To switch an iconic navigator to the list viewing mode, execute **View by Name** from the iconic navigator's popup menu.

The navigator displays the design objects as a list.

To switch a list navigator to the iconic viewing mode, execute **View by Icon** from the list navigator's popup menu.

The navigator displays the design objects as a matrix of large icons.

Figure 3-11. Iconic and List Navigators



Related Topics

[Search For a Design Object](#)

Search For a Design Object

After you open a navigator, you can use it to search for design objects on which to operate.

Navigation Buttons	159
Finding a Design Object in the Current Navigator Display	161

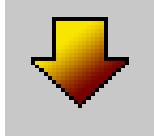
Navigation Buttons

Each navigator window's bottom frame contains five buttons with arrows on them.

You use these buttons to navigate to different destinations in the file system. To activate these buttons, you click them with the Select mouse button. These navigation buttons perform the following actions:

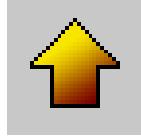
Table 3-1. Navigation Buttons

Explore Contents. Navigates into the selected directory and displays the design objects that are contained in that directory.



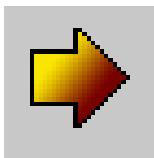
When the contents of a directory are displayed in the navigator, you are in *contents mode*. You are always in contents mode, unless you have just explored the references of a design object. After you navigate into the selected directory, the navigator adds that directory's name to the title bar. If you have set navigator filters to exclude objects from the navigator display, those objects that meet the filter requirements for exclusion are not visible when you explore the contents of a directory.

Explore Parent. If you are in *contents mode*, this button explores the container that contains the directory displayed in the navigator title bar. That is, this button navigates to the parent of the current design object's parent, and displays its contents. If you are in *reference mode*, this button explores the container of the selected reference's target object. That is, this button navigates to the parent of the object that is selected. After exploring the parent, the title bar displays the new location.



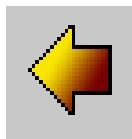
Clicking the **Explore Parent** button in contents mode, executes the `$explore_parent()` function. Clicking the **Explore Parent** button in reference mode, executes the `$explore_reference_parent()` function.

Explore References. Replaces the current display with the references of the selected design object. Reference pathnames are displayed exactly as they are stored in the attribute file. Exploring the references of a design object puts you in *reference mode*.



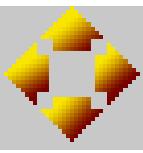
When you explore a design object's references, the navigator adds that object's name to the title bar and follows the name with a "@", indicating reference navigation. After exploring a design object's references, you can select another object in the display and explore its references or contents. When you alternate between exploring contents and references, the title bar reflects this by displaying both "/" and "@" in the pathname, signifying either a content or reference exploration.

Explore Back to Parent. Navigates back to the design object that holds the references currently displayed. That is, this button navigates back to the design object from which you originally explored references. This button is only activated when you are in reference mode; that is, when you have selected an object and explored its references. When you are in reference mode, an “@” is the right-most character in the title bar.



Clicking the **Explore Back to Parent** button in reference mode, executes the `$explore_parent()` function.

Go To. Displays a dialog box into which you can enter a soft or hard pathname to represent the file system destination to which you want to navigate. This destination must be a directory. After you enter the pathname of the destination directory and execute the dialog box, the navigator goes to that directory and displays its contents or the contents specified by your navigator filters, and the title bar displays the new location.



Related Topics

[\\$explore_parent\(\)](#)

Finding a Design Object in the Current Navigator Display

The Pyxis Project Manager navigators cannot always display all objects at the current containment or reference level.

The list navigator displays as many objects as it can, and you must scroll down to view the rest. The iconic navigator also displays as many objects as it can, based on the width of the window, and you must scroll down to view the rest of the icons.

To help you quickly find a specific design object in the current navigator, even if that object is not currently visible, the Pyxis Project Manager provides a search command. You search for an object by specifying its name or part of its name. In an iconic navigator, the Pyxis Project Manager places the cursor on the first object that matches the name you entered. In a list navigator, the Pyxis Project Manager scrolls the list, placing the object matching the search at the top of the visible window.

Note



If your search is unsuccessful, you should verify that you have not excluded the object from the navigator by setting navigator filters. The Search command cannot locate those design objects that exist in the current directory but are invisible because of navigation filters that you specified.

Procedure

1. Choose the **Explore > Search** menu item from a navigator popup menu, which displays a prompt bar.
Alternately, execute **File > Explore > Search** from the navigator's pulldown menu.
2. In the Pattern field, enter a unique part of the name of the object for which you are searching.
3. Press the Return key, or click on **OK**.

In an iconic navigator, a cursor moves to the first object that matches the name that you entered. In a list navigator, the located design object scrolls to the top of the window.

When you specify the name for which to search, you can use wild cards. The Search command uses the same “wild carding” scheme as the one used by UNIX System V. The Pyxis Project Manager finds and highlights the first item that matches the wild card’s criteria. For example, the pattern `*.lib` matches every object's name whose name ends with the string “.lib”.

Related Topics

[Regular Expressions](#)

Tool Invocation

The Pyxis Project Manager provides two basic methods of tool invocation:

- Create new data by invoking a tool from the tools window
- Edit existing data by navigating to the design object and selecting a tool with which to edit the object.

Invoking from the Tools Window	162
Invoking with the MGC_INVOKE_SETUP Variable	163
Invoking from a Navigator	165
Getting Information About a Tool	165
Changing the Toolbox Search Path	166

Invoking from the Tools Window

The tools window displays the tool viewpoints (tools) that you can invoke from your Pyxis Project Manager session.

To populate the tools window, the Pyxis Project Manager uses the toolbox search path.

Procedure

1. Choose **Open Tools Window** from the session window popup menu.

Alternately, execute **Windows > Open Tools Window** from the session window's pulldown menu.

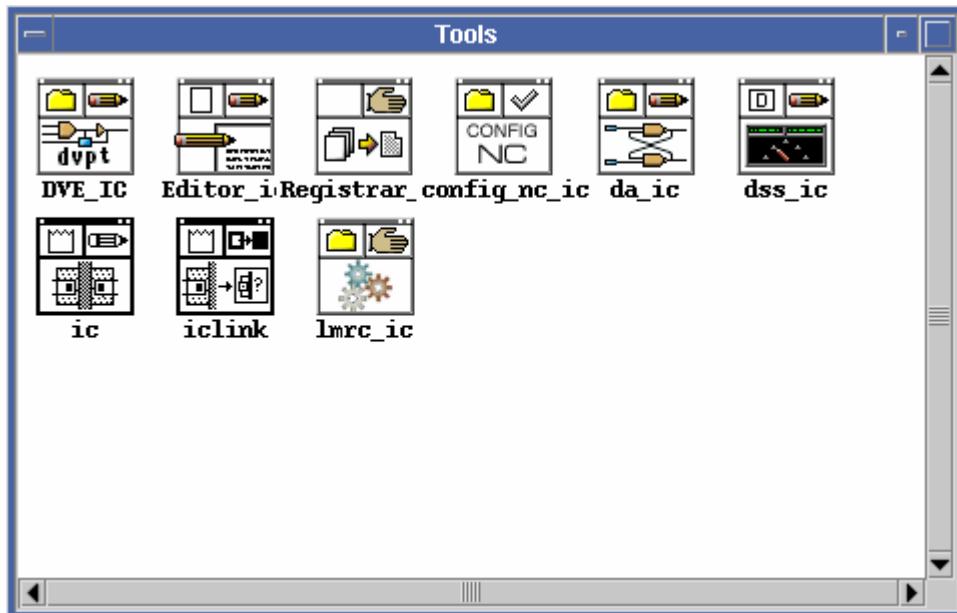
The tools window appears, displaying the tools that you can currently invoke. Figure 3-12 shows an example of a tools window.

2. Double-click the icon of the tool you want to invoke.

A dialog box may appear in which you choose the appropriate arguments for the tool invocation.

3. Execute the tool invocation dialog box. The tool that you selected invokes, creating its own session, independent of the Pyxis Project Manager. At this point, you can work with the invoked tool or return to the Pyxis Project Manager.

Figure 3-12. Example of a Tools Window



Related Topics

[Changing the Toolbox Search Path](#)

Invoking with the MGC_INVOKE_SETUP Variable

In order to specify a list of AMPLE files to be executed upon invoking any tool from Pyxis Project Manager, use the environment variable “MGC_INVOKE_SETUP” that contains a list of “:” separated paths to AMPLE files.

The paths may contain the following keywords:

- `<__MGC_CUR_PROJ_PATH__>` is the path to the currently active project in Pyxis Project Manager.
- `<__MGC_CUR_TECH_INVOKE__>` is the AMPLE file for the technology configuration of the currently active project.

Technology configurations can specify the location of the AMPLE file to run. An example location would be: `$TECH_LIB/userware/file1.ample`.

Following is an example of the `MGC_INVOKE_SETUP` environment variable:

```
MGC_INVOKE_SETUP=<__MGC_CUR_PROJ_PATH__>file1.ample:<__MGC_CUR_TECH_INVOKE__>:/var/tmp/foo.ample
```

The AMPLE files are executed from right to left; so the precedence is automatically set higher for files specified earlier in the list.

In the above example, the order of precedence and execution are as follows:

File	Executed	Precedence
/var/tmp/foo.ample	1st	3rd
<__MGC_CUR_TECH_INVOKE__>	2nd	2nd
<__MGC_CUR_PROJ_PATH__>/file1.ample	3rd	1st

Within these AMPLE files, you may queue a list of environment variables to pass to the invoked tool. This queue is used only for the next invocation, but it does not carry over into the following invocation, nor does it modify the current environment.

The AMPLE functions associated with the `MGC_INVOKE_SETUP` environment variable are:

- `$set_next_tool_env()` — enables you to add or edit an environment variable for the invoked tool.
- `$unset_next_tool_env()` — enables you to remove an environment variable for the invoked tool.
- `$get_next_tool_env()` — enables you to obtain the value for an environment variable for the invoked tool.

 **Note** If `MGC_INVOKE_SETUP` is not set, then by default, only the AMPLE file for the technology configuration of the currently active project is executed, provided the configuration has the AMPLE file defined.

Related Topics

[\\$set_next_tool_env\(\)](#)

[\\$unset_next_tool_env\(\)](#)

[\\$get_next_tool_env\(\)](#)

Invoking from a Navigator

Invoke any tool from the navigator.

Procedure

1. Navigate to the desired design object.
2. Open the design object, by selecting the design object and executing **Open** from the navigator's popup menu.

That menu cascades to a list of tools that can operate on the selected object.

3. Choose one of those tools.

When you select a tool, a dialog box may appear in which you choose the appropriate arguments for tool invocation.

4. Execute the dialog box, by clicking **OK**.

The tool invokes on your selected design object.

Related Topics

[Tool Invocation](#)

Getting Information About a Tool

In the tools window, you can get information about a tool's characteristics and current status.

Procedure

1. Select the object about which you want information.
2. Execute **Report Tool Info** from the tools window's popup menu. Alternately, execute **Report > Tool Info** from the tools window's pulldown menu.

A read-only window appears, displaying various items. Because the tools in the tools window are actually tool viewpoints, which are design objects, the information that appears in the window is the same as for any design object.

Related Topics

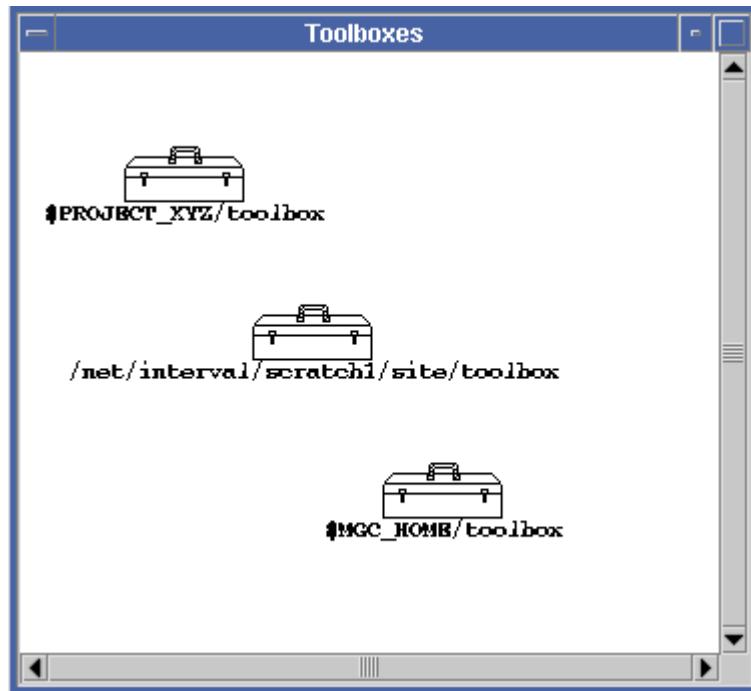
[Getting Information about a Design Object](#)

Changing the Toolbox Search Path

The toolbox window is closely related to the tools window.

The toolbox window displays toolboxes, which are actually file system directories. These toolboxes contain tool viewpoints. Figure 3-13 shows an example of a toolbox window.

Figure 3-13. Example of Toolbox Window



When populating the tools window, the Pyxis Project Manager searches the toolboxes displayed in the toolbox window, from the upper left to the lower right toolbox. When it finds a uniquely named tool viewpoint, the Pyxis Project Manager places that tool viewpoint in the tools window, making it available for invocation.

The order in which the Pyxis Project Manager searches the toolboxes is called the toolbox search path. By changing the toolbox search path, you can change which tools are available for invocation from the tools window.

When you change your toolbox search path and want to save those changes, you must explicitly save it using the toolbox popup menu item **Save** or the pulldown menu item **Edit > Save**. Saving the toolbox search path, by one of these methods, saves the current toolbox search path to the startup file `your_home/mgc/startup/dmgr_toolbox_path.startup`, where `your_home`

specifies your home directory. As a result, when you open the Pyxis Project Manager again, it uses this toolbox search path when populating the tools window.

If you have modified *dmgr_toolbox_path.startup* without having permission to write to this file, attempting to close your session causes the Pyxis Project Manager to display an error, and the session does not close. If this happens, see your system administrator.

Adding a Toolbox	167
Removing a Toolbox.....	168
Rearranging the Toolbox Search Path	168

Adding a Toolbox

When you add a toolbox to the toolbox search path, the Pyxis Project Manager searches the additional toolbox when populating the tools window.

As a result, the tools window displays (and you can invoke) any uniquely named tool viewpoints found in the new toolbox.

Because adding a toolbox always places it at the end of the toolbox search path your new toolbox may contain tool viewpoints whose name the Pyxis Project Manager has already found. As a result, they do not appear in the tools window. To place these hidden tool viewpoints in the tools window, you need to rearrange the toolbox search path so that your newly added toolbox is searched earlier.

Procedure

1. Execute the **View toolboxes** menu item on the tools window popup menu, which displays the toolbox window.
2. Execute **Add Toolbox** from the popup menu. This brings up a prompt bar.
3. Enter nothing at this prompt.
4. Enter nothing at the prompt and choose **File > Object Browser** from the toolbox window's pulldown menu. An object browser dialog appears, and you can navigate to and select a toolbox. When you find the toolbox that you want to add, select it and execute the object browser dialog.

After you specify the new toolbox, the toolbox appears in the lower-right corner of the toolbox window. As a result of adding the new toolbox to the toolbox search path, the Pyxis Project Manager searches this newly added toolbox last when populating the tools window.

If you now switch to tools mode, any uniquely named tool viewpoints found in your additional toolbox appear in the tools window.

Related Topics

[Removing a Toolbox](#)

[Rearranging the Toolbox Search Path](#)

Removing a Toolbox

As with adding a toolbox, removing a toolbox also changes the toolbox search path.

When you remove a toolbox, the Pyxis Project Manager no longer searches it for tool viewpoints when populating the tools window. This can have two effects: first, any uniquely named tool viewpoints that were in the removed toolbox no longer appear in the tools window; secondly, tool viewpoints that the removed toolbox *masked* by being searched first now appear in the tools window.

Procedure

1. In a toolbox window, select the toolbox that you want to remove.
2. Choose **Remove Toolbox** from the toolbox window's popup menu.

If you switch to tool mode, any uniquely named tool viewpoints in the toolbox you removed no longer appear and any masked tool viewpoints now appear.

Related Topics

[Adding a Toolbox](#)

[Rearranging the Toolbox Search Path](#)

Rearranging the Toolbox Search Path

You can rearrange the toolbox search path by dragging toolboxes around in the toolbox window using the mouse.

The position of the toolbox in relation to the other toolboxes determines its precedence. When populating the tools window, the Pyxis Project Manager searches toolboxes from upper left to lower right. For example, if you want a particular toolbox searched first, you place it in the upper left corner. All tool viewpoints in the upper left toolbox are displayed in the tools window. The tool viewpoints in the other toolboxes are displayed only if their names differ from those in the toolboxes previously searched.

Procedure

1. Select and drag the toolbox that you want searched first to the upper left-hand corner of the toolbox window.
2. Release the mouse button.

The toolbox window updates its display, neatly ordering the toolboxes as you specified.

To save your toolbox search path after you have modified it, you must execute either the toolbox popup menu item **Save** or the pulldown menu item **Edit > Save**.

Related Topics

[Adding a Toolbox](#)

[Removing a Toolbox](#)

Design Objects

The design object is the basic unit of data of all Mentor Graphics designs. Working with design objects between application sessions is one of the most important tasks that you do in the Pyxis Project Manager.

In fact, each of the three most important features of the Pyxis Project Manager, design navigation, customized tool invocation, and release configuration management, involves design objects.

The following text describes the navigator's popup menu, which provides most of the operations that you need to work with design objects, and describe the operations that you can perform on design objects.

Opening a Design Object	169
Getting Information about a Design Object	170
Copy, Move, and Rename a Design Object	171
Delete a Design Object	187
Change the References of a Design Object.....	190
Control Access to a Design Object.....	190
Creating a Basic Container Design Object	193

Opening a Design Object

In the Pyxis Project Manager navigator, you can open all design objects.

When you open a design object, the Pyxis Project Manager invokes a tool on it or performs some other operation. This choice depends on the design object's type. In a navigator, you can open a design object by selecting the object and choosing **Open >** from the navigator's popup menu.

Procedure

The following list explains how to open each type of design object:

- To open a *design object*, select the design object and choose **Open >** from the navigator's popup menu. This item cascades to a list of tools that can operate on the selected design object. If you choose one of the displayed tools, that tool invokes with the selected design object as input.

- To open a *tool viewpoint* in the navigator, select the tool viewpoint and choose the navigator popup's **Open > menu item**. This item cascades to the name of the tool and the tool invokes.
- To open a *container*, you can invoke a tool on it by using the method for opening design objects described previously, or you can explore its contents. To explore the container's contents, select the container and click the "down arrow" navigation button that appears on the bottom of the navigator window's frame, or double-click the container's icon. The navigator navigates into the container and displays its contents.
- To open a *directory*, double-click its icon; or select it and choose **Open > Explore Contents** from the navigator's popup menu; or select it and click the **Explore Contents** navigation button. In all of these cases, the navigator explores the contents of the selected directory.
- To open a *file*, select the file and choose **Open > Default Editor** from the navigator's popup menu. The Pyxis Project Manager invokes the default file editor on the selected file.

Alternatively, to open the default file editor on the file in read-only mode, select the file and choose **Open > Read-Only Editor** from the navigator's popup menu.

Related Topics

[Tool Viewpoints](#)

[Specifying the Default File Editor](#)

Getting Information about a Design Object

In a navigator, you can get information about a design object's characteristics and current status.

Procedure

1. Select that object and execute **Report > Object Info** from the navigator's popup or pulldown menu. A read-only window titled "Object Information Report" appears, displaying the following information:
 - **Object Name.** The name of the design object about which information is reported.
 - **Object Type.** The design object's type.
 - **Location.** The design object's absolute pathname.
 - **Protection.** The design object's protection status. In this string, "r" means read, "w" means write, "x" means execute, and "-" means no permission. From left to right, the first three characters denote your permissions, the second three denote your group's permissions, and the last three denote all others' permissions. These are the standard UNIX permissions.

- **Lock Status.** The design object is either locked or unlocked. The lock status may indicate a write lock, in which case, you cannot access the design object; or a read lock, in which case, you can read but not edit the design object.
- A read lock prevents others from editing the design object. However, others can still read-lock the design object. A write lock prevents others from locking or editing the design object. However, others can still read the object.
- **Released Status.** The design object is either released or un-released. If the design object is a released, versioned object, it cannot be opened for editing. An object's release status is independent of read and write locks, and unlike locks, is a permanent state.
- **Date Last Modified.** The date that the object was last edited.
- **Current Version Number.** The number of the design object's current version. This item appears only for versioned design objects.
- **Version Depth.** The design object's version depth. For versioned objects, this item indicates how many versions a design object keeps of itself before discarding the oldest version. A value of -1 signifies that there is an infinite version depth; versions are never be pruned off. This item appears only for versioned design objects.
- **Object Properties.** The design object's properties, if any exist. The properties are shown in the following format:
 - name: "value"These properties are attached to the entire design object and accumulate as the object evolves.
- **Fileset Members.** The design object's fileset members. Any missing required fileset members are reported.

Related Topics

[Changing an Object's Protection](#)

[Properties](#)

[Metadata Files](#)

[Design Object Filesets](#)

[Versions](#)

Copy, Move, and Rename a Design Object

The Pyxis Project Manager provides you with copy, move, and rename operations with which to manage your design objects.

When you perform these operations with Pyxis Project Manager, the data type being manipulated may contain properties that cause other functions to be called in order to maintain data integrity.

Copying a Design Object	172
Copying with the Popup or Pulldown Menu	173
Copying by Dragging	176
Copying a Previous Version	176
Releasing a Design Object	177
Releasing a Design Object using the Command Prompt	177
Releasing a Design Object Using Popup or Pulldown Menu	179
Moving a Design Object	182
Moving with the Popup or Pulldown Menu	183
Moving by Dragging	185
Renaming a Design Object	186

Copying a Design Object

Copying duplicates one or more selected design objects, leaving the selected design objects intact and placing duplicates in the specified destination directory.

When you copy design objects that refer to other design objects which are being copied in the same operation, references between them are updated to reflect their new location.

Automatic reference updating occurs only when you copy the related design objects in the same operation. For example, if design objects A and B reference each other, to copy them to a new location and update their references to reflect that location, you must copy them both in the same operation. If you copy A in one operation, and then copy B in a second operation, the copy of A references the original B, and the copy of B references the original A.

You can copy design objects in the Pyxis Project Manager in the following ways:

- By executing the **Copy Object** command, which you can access through the popup menu option **Edit > Copy**, or by typing the command in a popup command line.
- By executing **Edit > Copy** from the navigator's pulldown menu, and performing the move operation using the Move Object command,
- By executing **MGC > Design Management > Copy object** from the navigator's pulldown menu. By dragging the design object's icon from one iconic navigator to another.



Note

Although the following features are described in terms of copying only a single design object, they also apply when you select and copy multiple design objects.

The **Copy Object** command provides the following additional features:

- The copy operation can be interrupted. When you copy a design object, all objects that it contains are also copied. The time needed to completely copy a single design object can be deceiving, if the object has a very large containment network. In a case like this, you

can halt the execution of the copy operation by pressing the Ctrl and Backslash keys together.

When you halt the copy operation, a dialog box is displayed prompting you about whether the operation should halt or continue. If you choose to halt the operation, the operation is aborted and the standard clean-up procedure is performed.

- When copying a single design object, you can specify a new leaf name for the object instead of a destination directory. If you specify a leaf, the copied object is added in the active navigator's current directory.

If you have only one design object selected, and if you accidentally specify a non-existent directory for the target, the Pyxis Project Manager copies the selected object, creating a new object with the name of the non-existent directory that you specified.

When you select multiple objects for copy, you must always specify an existing directory for the destination, and each copied object keeps its current leaf name.

Related Topics

[Copy Design Object](#)

[Copying Design Objects](#)

Copying with the Popup or Pulldown Menu

Copy a design object by using the menu.

Procedure

- In a navigator, select the objects you want to copy.
- To copy a design object using the command prompt bar, follow these steps:

Hit the Spacebar key to invoke the command prompt. In the command prompt, type the Copy Object command to bring up the COPy OBject prompt bar.

or

Click the mouse Menu (right) button, or press the F4 function key to bring up the navigator's popup menu. Choose the **Edit > Copy** option.

or

From the navigator's pulldown menu, choose the **Edit > Copy** option

- In the prompt bar's Destination field, enter the directory into which you want to copy the selected objects.
 - If the directory exists, the selected design objects are copied into that directory.

- b. If the directory does not exist and if multiple design objects are selected, an error message appears.
- c. If the directory does not exist and if a single design object is selected, the Copy Object command simply copies the selected object to that name. However, this works only if everything in the destination pathname exists except the leaf.

Instead of specifying the destination as a full pathname, you can specify the destination as only a leaf name. If you specify a leaf, the copied object is added in the active navigator's current directory. For example, to copy design object *tom* to the name *phil* in the same directory, specify "phil" in the "Destination" field.

If you have only one design object selected, and if you accidentally specify a non-existent directory for the target, the Pyxis Project Manager copies the selected object, creating a new object with the name of the non-existent directory that you specified.

When you select multiple objects for copy, you must always specify an existing directory for the destination, and each copied object keeps its current leaf name.

4. Specify the behavior of the copy operation, by pressing the prompt bar's **Options** button. This brings up the Copy Object Options dialog box, as shown in [Figure 3-14](#).

Figure 3-14. Copy Object Options dialog box



5. The options have the following meaning:

- **Library Filter**

With this option, you specify which objects to exclude from the copy, based on their pathname. You can specify the Path Pattern by entering the literal string or by using UNIX System V regular expressions. The default is to include all objects in the release.

- **Follow References**

With this option, you specify whether reference traversal is turned on or off. Reference traversal determines whether externally referenced design objects of the

selected design objects are included in the copy. By default, containment traversal is always set to on, which means that a design object's contained objects are included in the copy.

- **Update References**

With this option, you specify whether the Pyxis Project Manager updates the references of the copied object after the copy operation is complete. By default, the Pyxis Project Manager always updates references.

- **Preview Only**

With this option, you specify whether this function issues a preview report of the expected results of this operation, or executes the copy operation.

- **Copy Mode**

With this option, you specify whether to copy the current version of the selected design objects, or the entire object.

- **Overwrite Conflicts**

With this option, you specify the action that the Copy Object command takes when a design object already exists in the destination with the same name as a source design object.

- i. **Cancel**

Cancels the entire copy operation if any name conflicts are found at the destination.

- ii. **Replace**

Replaces conflicting non-container type design objects with copies of the selected design objects.

- iii. **Ask**

For each selected design object, if a naming conflict is found, the Copy Object command asks if you want to leave the destination object intact or to replace it with a copy. If an overwrite conflict occurs and an action is not specified, this is the default.

6. Execute the Copy Object Options dialog box, after selecting your copy options. The dialog box disappears, and you return to the Copy Object prompt bar.
7. Execute the Copy Object prompt bar by pressing the Return key or clicking the **OK** button.

Depending on the overwrite conflicts that you specified in step 4, the selected design objects are either copied to the destination that you specified, or the copy operation is

canceled, or Pyxis Project Manager asks you if you want to replace existing design objects.

Related Topics

[\\$get_navigator_directory\(\)](#)

[\\$get_navigator_directory_hard\(\)](#)

Copying by Dragging

Copy a design object by dragging the icon.

Procedure

1. Open two navigators.
2. Navigate to the directory that contains the design objects that you want to copy. In the second navigator, go to the destination directory where you want to place the copied object.
3. Set the navigator that contains the source design object to iconic mode, by executing **Open Navigator > View by Icon** from the navigator's popup menu, or **Windows > Open Navigator > View by Icon** from the navigator's pulldown menu.
4. Select the design objects that you want to copy, in the navigator that contains them.
5. Press the Shift key. While keeping it depressed, drag one of the selected icons from the source navigator into the navigator that is rooted at the destination.
6. When one of the transparent icon images is in the destination navigator, release the mouse button.

The icons of the copied design objects appear in the destination directory, indicating that the copy is complete.

Note



Copying an object by dragging its icon copies all the versions of the selected design object. The Pyxis Project Manager asks you if you want to replace conflicting items, if any conflicting items are found.

Related Topics

[\\$open_navigator\(\)](#)

Copying a Previous Version

In the version window, you can select and copy a previous version of a design object, using the Copy Version command.

The copied version becomes version number 1 of a new design object.

Related Topics

[Copying a Version](#)

Releasing a Design Object

Releasing a design object creates a protected copy of the design object or set of design objects, in a specified destination directory.

The process of releasing a design object is a simplified alternative to the process of releasing a configuration. When you release a configuration, you must create a configuration object, specify complex build rules, and build the configuration. Although, this release method is valuable and necessary at times in the design process, you do not always need this complexity.

When you perform a simplified release on a selected set of design objects, the current version of each object is released to the destination directory, unless the object is referenced as a fixed version. If the object is referenced as a fixed version, the specified version is included in the release.

You can release objects from the navigator window or from the toolkit without performing these additional tasks by issuing the **Release Object** command.

Related Topics

[Release Design Object](#)

`$release_object()`

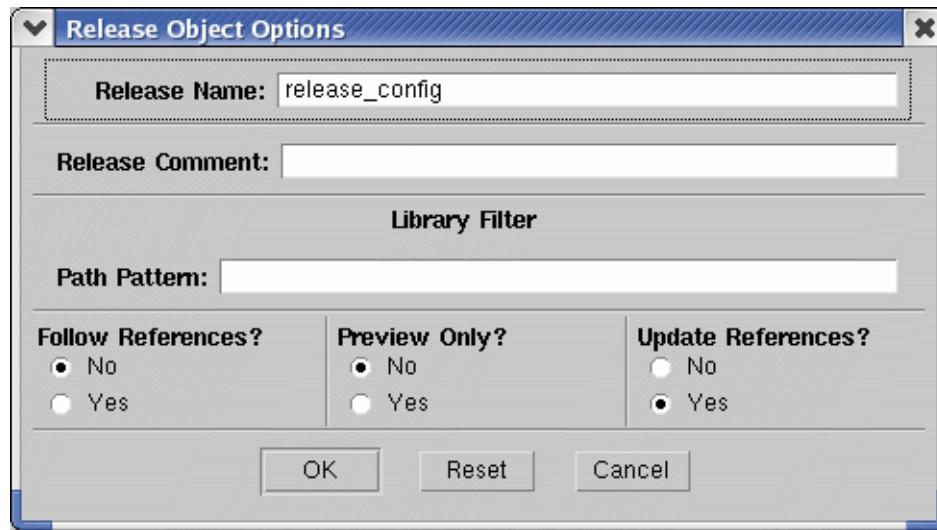
Releasing a Design Object using the Command Prompt

Release a design object using the command prompt.

Procedure

1. Hit the space bar key to invoke the command prompt.
2. Type the **Release Object** command to bring up the RELease OBject prompt bar.
3. Type in the destination path where the object is to be released in the Target field.
4. Alternately, click the Options button and bring up the **Release Object Options** dialog box, as shown in [Figure 3-15](#).

Figure 3-15. Release Object Options Dialog Box



5. The options have the following meaning:

- Release Name

With this option, you specify the name of the new configuration object in the target directory. If you do not specify a name, the default name “release_config” is assigned to the configuration object.

- Release Comment

With this option, you specify any annotations that you want to associate with the released object. For example, you may want to specify the comment string “Released by: John Smith” to identify that you performed the release operation.

You can read comments attached to any object by doing the following:

- a. Select the release configuration object

- b. Invoke the menu item Report > Show Versions. A window showing the versions appears. Select the version you want information on.

- c. Invoke the menu item Report > Version Info. Pyxis Project Manager then brings up a window displaying version information, including any comments.

- Library Filter

With this option, you specify which objects to exclude from the release, based on their pathname. You can specify the Path Pattern field by entering the literal string or by using UNIX System V regular expressions. The default is to include all objects in the release.

- Follow references

With this option, you specify whether reference traversal is turned on or off. Reference traversal determines whether externally referenced design objects of the selected design objects are included in the release. By default, containment traversal is always set to on, which means that a design object's contained objects are included in the release.

- Preview only

With this option, you specify whether this function issues a preview report of the expected results of this operation to the session configuration area, or executes the release operation.

- Update references

With this option, you specify whether references are updated to reflect the location of the released copy of the object.

Related Topics

[Release Design Object](#)

`$release_object()`

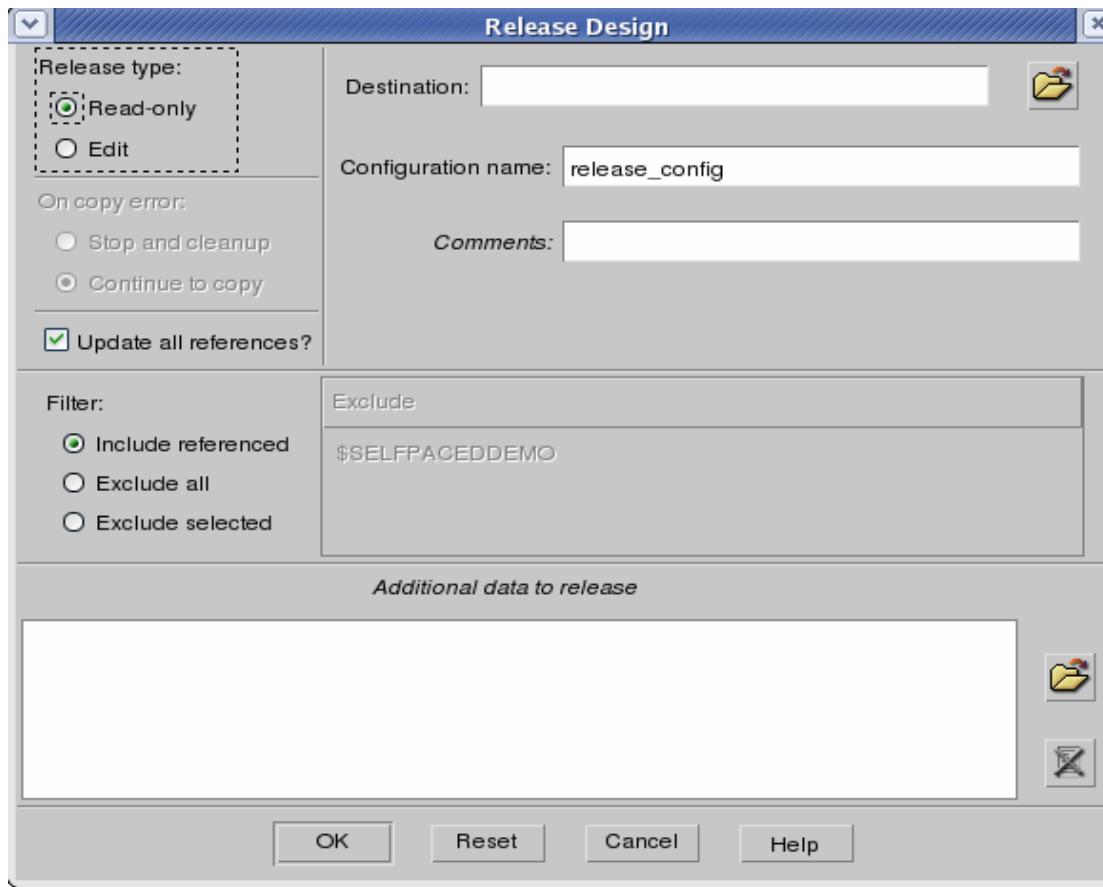
Releasing a Design Object Using Popup or Pulldown Menu

Release a design object using the navigator popup menu or a pulldown menu.

Procedure

1. In a navigator, select the design objects that you want to release.
2. Click the mouse Menu (right) button in the navigator window, or execute **Release > Object** from the navigator's popup menu. Alternately, execute **File > Release** from the navigator's pulldown menu.
3. Any of the above navigation methods brings up the Release Design dialog box, as shown in [Figure 3-16](#).

Figure 3-16. Release Design Dialog Box



The options have the following meaning:

- Destination

With this option, you specify the name of the new configuration object in the target directory. If you do not specify a name, the default name “release_config” is assigned to the configuration object.

- Configuration name

The default name is “release_config”. Specify the new name for the configuration in this field, in order to change the default value.

- Comments (optional)

With this option, you specify any annotations that you want to associate with the released object. For example, you may want to specify the comment string “Released by: John Smith” to identify that you performed the release operation.

You can read comments attached to any object by doing the following:

- a. Select the release configuration object.

- b. Invoke the menu item Report > Show Versions. A window showing the versions appears.
 - c. Select the version you want information on.
 - d. Invoke the menu item Report > Version Info. Pyxis Project Manager then brings up a window displaying version information, including any comments.
 - Release type
Select the “Read-only” or “edit” button, depending on whether you want the release object to be edited or not. The default option is the “Read-only” type.
 - On copy error
Select the “Stop and cleanup” or “Continue to copy” button, to specify what the Release Design operation needs to do, when it encounters a copy error. The default is the “Continue to copy” option.
 - Update all references
With this checkbox, you specify whether or not all references are updated to reflect the location of the released copy of the object. By default, this option is checked.
 - Library symbol filtering
With this option, you specify which library symbols are to be included or not. Select one of these options: Include referenced, Exclude all, or Exclude selected. To specify the select libraries to exclude, use the “Select libraries to exclude” field. The default is to include all referenced libraries in the release.
 - Select libraries to exclude
With this option, specify which libraries to exclude from the release, based on their soft prefixes.
 - Additional data to release (optional)
With this option, you can browse for any additional data objects to add along with this new release design.
4. Execute the Release Design dialog box by pressing the Return key or clicking the **OK** button.
- The selected design objects are released to the destination that you specified.

Related Topics

[\\$get_navigator_directory\(\)](#)

[\\$release_object\(\)](#)

[\\$get_navigator_directory_hard\(\)](#)

Moving a Design Object

The Moving task relocates one or more design objects to the specified destination directory.

When you move design objects that refer to other moved design objects, references between them are updated to reflect their new location.

Automatic reference updating occurs only when you move the related design objects in the same operation. For example, if design objects A and B reference each other, to move them to a new location and update their references to reflect that location, you must move them both in the same operation. If you move A in one operation and move B in a second operation, in the destination directory, A references the original B (which no longer exists), and B references the original A (which no longer exists).

In Pyxis Project Manager, you can move a design object in the following ways:

- By executing the Move Object command, which you can access through a popup menu or by typing the command in a popup command line.
- By executing **Edit > Move** from the navigator's pulldown menu, and performing the move operation using the Move Object command,
- By executing **MGC > Design Management > Move object** from the navigator's pulldown menu.
- By dragging the design object's icon from one iconic navigator to another.

The Move Object command provides the following features:

- Moving a design object in Pyxis Project Manager always moves all versions of the selected object.
- Occasionally, you may specify a destination for a move that conflicts with an existing design object. To handle this case smoothly, the Move Object command lets you specify, prior to the move, how Pyxis Project Manager handles conflicts in the destination directory. You can specify that Pyxis Project Manager does one of the following:
 - Asks you if you want to continue
 - Cancels the entire move operation
 - Replaces the conflicting design object with the object being moved.
- The move operation can be interrupted. When you move a design object, all objects that it contains are also moved. The time needed to completely move a single design object can be deceiving, if the object has a very large containment and/or reference network. In a case like this, you can halt the execution of the move operation by pressing the Ctrl and Backslash keys together.

- When you halt the move operation, a dialog box is displayed prompting you about whether the operation should halt or continue. If you choose to halt the operation, the operation is aborted and the standard clean-up procedure is performed.
- When moving a single design object, you can specify a new leaf name for the object instead of a destination directory. If you specify a new leaf, the moved design object is added in the active navigator's current directory, which is, in effect, a "rename" operation.

If you have only one design object selected and you accidentally specify a non-existent directory for the target, the Pyxis Project Manager moves the selected object, creating a new object with the name of the non-existent directory that you specified.

- When you select multiple objects for move, you must always specify an existing directory for the destination, and each moved object keeps its current leaf name.

Related Topics

[Moving Design Objects](#)

Moving with the Popup or Pulldown Menu

Move a design object by using the menu.

Procedure

1. In a navigator, select the objects you want to move.
2. To move a design object using the command prompt bar, follow these steps:
 - Hit the Spacebar key to invoke the command prompt. In the command prompt, type the Move Object command to bring up the MOVe OBject prompt bar.

or

Click the mouse Menu (right) button, or press the F4 function key to bring up the navigator's popup menu. Choose the **Edit > Move** option.

or

From the navigator's pulldown menu, choose the **Edit > Move** option.

Note



Although the following features are described in terms of moving only a single design object, they also apply when you select and move multiple design objects.

3. In the prompt bar's Destination field, enter the directory into which you want to move the selected objects:

- a. If the directory exists, the selected design objects are moved into that directory.
- b. If the directory does not exist and if multiple design objects are selected, an error message appears.
- c. If the directory does not exist and if a single design object is selected, the Move Object command moves the selected object to that name, whether the selected object is a directory or not. However, this works only if everything in the destination pathname exists except the leaf.

Instead of specifying the destination as a full pathname, you can specify the destination as only a leaf name. If you specify a leaf, the moved design object is added in the active navigator's current directory. For example, to move design object *tom* to the name *phil* in the same directory, simply specify "phil" in the Destination field.

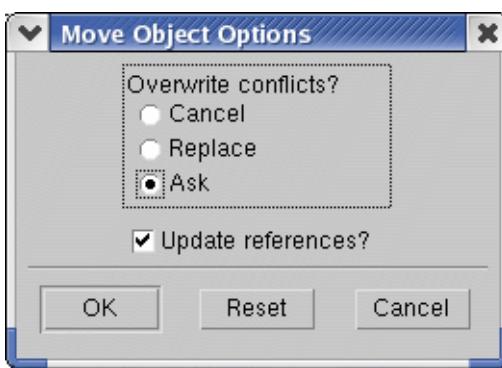
When moving a single design object, you can specify a new leaf name for the object instead of a destination directory. If you specify a new leaf, the moved design object is added in the active navigator's current directory, which is, in effect, a "rename" operation.

If you have only one design object selected and you accidentally specify a non-existent directory for the target, the Pyxis Project Manager moves the selected object, creating a new object with the name of the non-existent directory that you specified.

When you select multiple objects for move, you must always specify an existing directory for the destination, and each moved object keeps its current leaf name.

4. Press the prompt bar's **Options** button to display the Move Object Options dialog box, as shown in [Figure 3-17](#), that lets you specify how the Move Object command handles the overwriting of a design object.

Figure 3-17. Move Object Options Dialog Box



5. Select one of the options in the Overwrite Conflicts field:
 - **Cancel** — Cancels the entire move operation if any name conflicts are found at the destination.

- **Replace** — Replaces conflicting non-container type design objects with the selected design objects. Container type design objects are not replaced.
 - **Ask** — Asks if you want to leave the destination object intact or replace it with the selected design object. If an overwrite conflict occurs and an action is not specified, this is the default value.
6. Update References: With this checkbox, you specify whether or not all references are updated to reflect the location of the moved object. By default, this option is checked.
 7. Execute the dialog box and return to the Move Object prompt bar.
 8. Execute the Move Object prompt bar by pressing the Return key or clicking the **OK** button. Depending on the overwrite conflicts that you specified in step 3, the selected design objects are either moved to the destination that you specified, or the move operation is canceled, or Pyxis Project Manager asks you if you want to replace existing design objects.

Related Topics

[\\$get_navigator_directory\(\)](#)

[\\$get_navigator_directory_hard\(\)](#)

Moving by Dragging

Move a design object by dragging its icon.

Procedure

1. In one navigator, navigate to the directory that contains the design objects that you want to move. In another navigator, navigate to the directory where you want to move them.
2. Set the navigator that contains the source design object to iconic mode, by executing **Open Navigator > View by Icon** from the navigator's popup menu, or **Windows > Open Navigator > View by Icon** from the navigator's pulldown menu.
3. Drag one of the selected icons from the source navigator into the navigator that is rooted at the destination.
4. When one of the transparent icon images is in the destination navigator, release the mouse button.

The icons of all the selected design objects appear in the destination directory, indicating that the move is complete.

When you move by dragging, the Pyxis Project Manager asks you if you want to replace conflicting items.

Related Topics

[Moving Design Objects](#)

Renaming a Design Object

Renaming changes the name of a single design object.

In the Pyxis Project Manager, when you rename a container design object, the Pyxis Project Manager updates the container's references, and the references of any contained objects, to reflect the new name of the container.

Procedure

1. Select the design object whose name you want to change.
2. Execute **Edit > Change > Name** from the navigator popup or pulldown menu, which displays a prompt bar.
3. Specify the behavior of the renaming operation by pressing the prompt bar's **Options** button. The Change Object Name Options dialog box appears, as shown in [Figure 3-18](#).

Figure 3-18. Change Object Name Options Dialog Box



This lets you specify one of the following options for overwriting an object of the same name:

- **Cancels** the renaming operation if a design object exists with the same name.
 - **Replaces** the conflicting design object with the selected design object.
 - **Asks** if you want to leave the conflicting object intact or to replace it with the selected design object. If an action is not specified, this is the default value.
4. Execute the Change Object Name Options dialog box, which returns you to the prompt bar.
 5. Enter the object's new name in the dialog box's New Object Name field.
 6. Execute the Change Object Name prompt bar by pressing the Return key.

Depending on the overwrite conflicts that you specified in step 3, the selected design object is renamed to the name that you specified, the renaming operation is canceled, or the Pyxis Project Manager asks if you want to replace a conflicting design object.

Related Topics

[Design Objects](#)

Delete a Design Object

Deleting a design object in the Pyxis Project Manager, removes all versions of the selected design object from the current directory.

You can delete a single design object or multiple design objects in the same operation, by using either of the following two methods.

1. By executing the Delete Object command, which you can access through a popup menu or by typing the command in a popup command line. When you delete a design object using this method, the design object is permanently deleted and is irretrievable.
2. By dragging a design object's icon to the trash can. When you drag a design object to the trash can, you can retrieve the design object from the trash at any time, prior to ending your Pyxis Project Manager session. In a navigator, select the objects you want to move.

Deleting with the Menu	187
Deleting by Dragging to the Trash Can	188
Using the Trash Window	189
Emptying the Trash.....	190

Deleting with the Menu

Delete a design object using the command prompt bar.

Procedure

1. In a navigator, select the design objects that you want to delete.
2. Hit the space bar key to invoke the command prompt. In the command prompt, type the Delete Object command to bring up the DElete OBject prompt bar.

or

Click the mouse Menu (right) button, or press the F4 function key to bring up the navigator's popup menu. Choose the **Edit > Delete > Object** option.

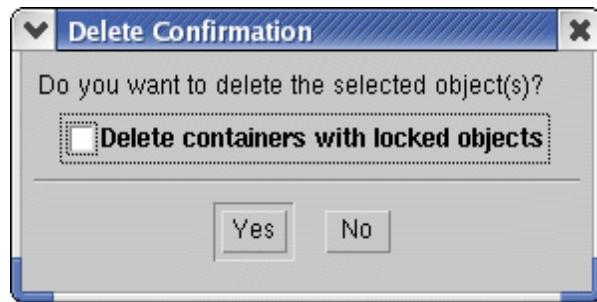
or

From the navigator's pulldown menu, choose the **Edit > Delete > Object** option.

3. This brings up the Delete Confirmation dialog box, as shown in [Figure 3-19](#), asking you to confirm or cancel the deletion.
4. In addition, a checkbox option to “Delete containers with locked objects” is displayed.
5. Confirm the deletion by clicking the **Yes** button.

The selected design objects are permanently deleted. You cannot retrieve them.

Figure 3-19. Delete Confirmation Dialog Box



Related Topics

[\\$delete_design_object\(\)](#)

Deleting by Dragging to the Trash Can

The trash can provides an alternative to permanent deletion.

The trash can icon, shown in [Figure 3-20](#), appears in the lower right corner of the Pyxis Project Manager session window.

The *trash window* displays the design objects you have placed in the trash can during the current session, as icons. At any time you can retrieve a design object that you placed in the trash can, by dragging that design object out of the trash window. Note, however, that selected design objects are permanently deleted when you empty the trash can.

Figure 3-20. Trash Can Icon



Each session of the Pyxis Project Manager has its own trash can. There are no conflicts when you have multiple Pyxis Project Managers open. In the event of a system crash, you can find the trash files in the directory `/tmp`; all `dme_trash_dir_x` directories (where `x` designates an integer) contain the trash files.

Procedure

1. In an iconic navigator, select the design objects that you want to delete.
2. Drag one of the selected icons from the navigator into the trash can, using the Select mouse button.
3. When one of the transparent icon images is sitting on the trash can, release the Select mouse button.

The trash can icon changes to appear overflowing with trash, indicating that the selected design objects are contained in the trash can.

Related Topics

[Using the Trash Window](#)

Using the Trash Window

As an alternative to dragging to the trash can, you can also drag objects directly to the trash window.

The trash window displays those objects that you have dragged to the trash can and allows you to retrieve a design object from the trash before the deletion is final. At any time, you can search the trash window for a desired object by using the **Search** menu item available in both the popup and pulldown menus.

Dragging an object to the trash is like moving the object to the trash window. In other words, the design object's references are automatically updated to reflect the new location: the trash window. This updating process is very important because when you retrieve the objects from the trash can, you may place them in a location that is different from their original location. Because references were updated when you moved these objects to the trash, they are updated again when you retrieve from the trash, and you can use the retrieved design objects without changing any references.

Procedure

1. In a navigator, go to the location in which you want to place the retrieved design objects.
2. Open the trash window by double-clicking the trash can icon.
3. In the trash window, select the design objects that you want to retrieve.
4. Execute the **Untrash Object** menu item from the trash window's popup menu.

A prompt bar is displayed requesting the destination directory for the objects which are being removed from the trash.

5. In the Destination field, enter the full directory pathname to which you want to move the selected objects.

6. Execute the prompt bar by clicking the **OK** button.

If you enter a valid directory pathname, the trash window and navigator are updated, indicating that the selected design objects are retrieved. If you enter an invalid directory pathname, the selected design objects remain visible in the trash window.

Related Topics

[Deleting by Dragging to the Trash Can](#)

[Emptying the Trash](#)

Emptying the Trash

After you drag objects to the trash can, you can empty the trash to make the deletions permanent.

Procedure

- The Pyxis Project Manager automatically empties the trash when you close the session.
- To empty the trash can, choose the session window's pulldown menu item **Edit > Empty Trash**.

Related Topics

[Using the Trash Window](#)

[Deleting by Dragging to the Trash Can](#)

Change the References of a Design Object

As mentioned in Chapter 2, moving and deleting design objects sometimes break the references that point to the moved or deleted objects.

For this reason, after you move or delete design objects, you frequently need to change the reference pathnames of other design objects. For this operation, the Pyxis Project Manager provides the Change Object References command.

Related Topics

[Changing Reference Pathnames](#)

Control Access to a Design Object

In the Pyxis Project Manager, each design object has its own *protection*.

A design object's protection refers to the granting of access permissions to other users. If you create a design object, you are the owner of that object and can set the access permissions for that object, giving or denying access to other users.

In the Pyxis Project Manager, design object protection is governed by UNIX System V ownership access permissions. In UNIX, ownership is denoted by *user*, the user or owner of the file; by *group*, the user's group; and by *other*, all others outside the user's group. UNIX permissions are denoted by *read (r)*, *write (w)*, and *execute (x)*.

The Pyxis Project Manager provides you with information about the ownership access permissions in a form similar to the UNIX **ls -l** command. Therefore, a permission string allowing a single owner to read, write, and execute would consist of the string **rwx**. Permissions for each of the three owner groups would therefore consist of three similar strings concatenated together, in the following order: *user* permissions, *group* permissions, *other* permissions. For example, the permission string **rwxrwxr-x** shows that the *user* and *group* have read, write, and execute permission, while the *other* only have read and execute permission.

Although the Pyxis Project Manager provides functions to set and support UNIX protections, these functions do not provide additional functionality or support for maintenance of these protections.

Showing an Object's Protection	191
Changing an Object's Protection.....	191

Showing an Object's Protection

Show a design object's protection information.

Procedure

1. In a navigator, select the design objects whose protection you want to show.
2. Choose **Report > Object Info** from the navigator's popup or pulldown menu.

The protection information for the selected object is displayed in a report window, under the *Protection* heading. If more than one object is selected, a corresponding number of report windows are displayed.

Related Topics

[Getting Information about a Design Object](#)

Changing an Object's Protection

You can change the protection of one or more selected objects from a navigator.

The Pyxis Project Manager provides you with the same functionality as the UNIX **chmod** command. However, you can change an object's protection only if you are the owner of the object.

Procedure

1. In a navigator, select the design object(s) whose protection you want to change.
2. Choose **Edit > Change > Protection** from the session window's pulldown menu.

A dialog box appears, as shown in Figure 3-21, displaying the design object's current protection and allowing you to click on the protection buttons to select new protection settings.

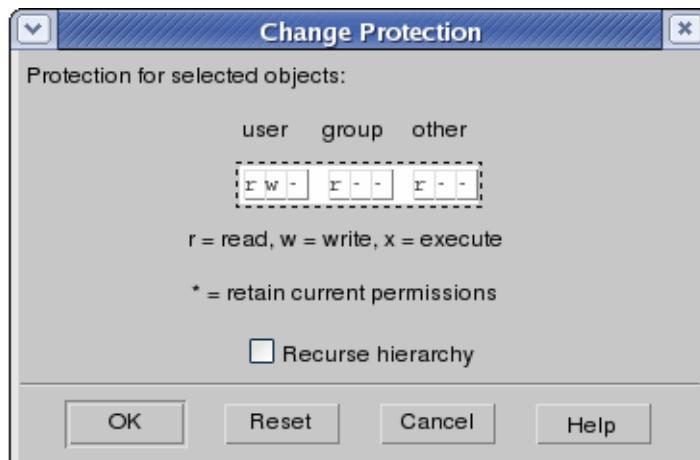
3. Select the protections that you want for each of the owner groups *user*, *group*, and *other*, by clicking on the permission buttons.

When you click on a permission button, the button display switches between the following settings:

- “r”, “w”, or “x”, denoting *read*, *write*, or *execute*
- “-”, denoting *deny access*
- “*”, denoting *retain current positions*

If more than one design object is selected, the dialog box shows all protections set to asterisks “*”. If you change a setting to something other than “*”, all of the selected design objects receive that setting. If you leave the “*” in a particular field, each selected design object keeps its own setting for that field.

Figure 3-21. Change Protection Dialog Box



In the following example, the Pyxis Project Manager changes the *user* and *group* protections of all three objects in the same way, while leaving the protection of *other* unchanged. Assume you have three objects selected, whose permissions are as follows:

rw-rw-rw- (object 1)
rwxrw-rw- (object 2)
rwxr---r--- (object 3)

You select **rwxrwxr***** in the dialog box. The Pyxis Project Manager changes the protections to the following:

rwxrwxrw- (object 1)

rwxrwxrw- (object 2)

rwxrwxr-- (object 3)

If you click the **Cancel** button at any time during this selection process, the original protection remains unchanged.

4. Click the **OK** button.

The object's protection changes to the settings you specified in step 3.

Related Topics

[\\$change_protection\(\)](#)

Creating a Basic Container Design Object

Depending upon which applications are installed in your Mentor Graphics tree, you can create *basic container* design objects while in the Pyxis Project Manager; basic containers are container design objects that have only one fileset member, a directory.

Containers are design objects whose fileset includes at least one directory and one or more additional fileset members. You use these containers strictly for organizing your designs. Unlike ordinary directories, they have unique icons, which identify their type. For example, you might be able to add a “file drawer” container, whose icon looks like a file drawer.

Procedure

1. In a navigator, choose **File > Add > Container** from the pulldown menu, which displays the Create Container dialog box.
2. In the Container path field, enter the pathname of the container that you want to create.
You can specify a full pathname or only a leaf name. If you specify a leaf, the container is added in the active navigator's current directory.
3. Select a container type from the scrollable Container Type list.
4. Execute the dialog box, by clicking **OK**.

A container with the appropriate icon is created. If you specified a leaf for Container path, the active navigator's window updates to display the new container.

Related Topics

[\\$add_container\(\)](#)

File System Objects

A file system object is a design object that allows you to access the file system while in the Pyxis Project Manager.

The Pyxis Project Manager displays two file system objects: file and directory.

Opening a File System Object	194
Adding a File	194
Adding a Directory	195
Adding a Container	195
Editing a File	196
Salvaging a Design Object	196

Opening a File System Object

In the Pyxis Project Manager navigator, you can open all file system objects by executing **Open >** from the navigator's popup menu.

You can also open container design objects and explore their contents by double-clicking the container's icon.

Procedure

- To open a directory, double-click its icon; or select it and choose **Open > Explore Contents** from the navigator's popup menu, or select it and click the **Explore Contents** navigation button, or select **File > Open > Explore Contents** from the navigator's pulldown menu. In all cases, the navigator explores the contents of the selected directory.
- To open a file, select it and choose **Open > Default Editor** or **Open > Read-Only Editor** from the navigator's popup menu. Alternately, execute **File > Open > Default Editor** or **File > Open > Read-Only Editor** from the navigator's pulldown menu. The Pyxis Project Manager invokes the specified file editor on the selected file.

Related Topics

[Specifying the Default File Editor](#)

Adding a File

The Pyxis Project Manager does not provide a command for you to create a new file.

Procedure

However, you can create a new file while in the Pyxis Project Manager in two ways:

- Open a tools window and double-click the *Editor* tool. If the default file editor is set to Notepad, an untitled Notepad appears. When you finish editing the new file, you can save it to a name by choosing **File > Save As** from the Notepad pulldown menu.
- Choose the **MGC > Notepad > New** menu item from the pulldown menu in any Pyxis Project Manager window.

For details about Notepad, refer to the *Notepad for Pyxis User's and Reference Manual* available on SupportNet.

Related Topics

[Specifying the Default File Editor](#)

Adding a Directory

Add a directory in the Pyxis Project Manager.

Procedure

1. In a navigator, choose **File > Add > Directory** from the pulldown menu, which displays a prompt bar.
2. Type in the name of the directory to be added.
You can specify a full pathname or only a leaf name. If you specify a leaf name, the directory is in the active navigator's current directory.
3. Execute the prompt bar by clicking **OK**. The directory that you specified is created. If you specified a leaf for the Directory Pathname, the active navigator's window updates to display the new directory.

Related Topics

[Specifying the Default File Editor](#)

[\\$add_directory\(\)](#)

Adding a Container

Add a basic container in the Pyxis Project Manager.

Procedure

1. In a navigator, choose **File > Add > Container** from the pulldown menu.

A dialog box called Create Container appears, in which you specify the name of the container to be added and its type. You can specify a full pathname or only a leaf name. If you specify a leaf, the container is added in the active navigator's current directory.

2. Execute the dialog box, by clicking **OK**.

The container with the name and type you specified is created. If you specified a leaf for the Container path field, the active navigator window updates to display the new container.

Related Topics

[Specifying the Default File Editor](#)

[\\$add_container\(\)](#)

Editing a File

Edit an existing file in Pyxis Project Manager.

Procedure

- Choose **MGC > Notepad > Open > Edit** from the pulldown menu of any Pyxis Project Manager window.
An object browser dialog appears with which you can navigate to a file, select it, and execute the dialog box. Notepad invokes with the selected file as input.
- Open a tools window and double-click the *Editor* tool.
When you execute the prompt bar, the default file editor invokes, with the specified file as input.

Related Topics

[Specifying the Default File Editor](#)

[\\$edit_file\(\)](#)

Salvaging a Design Object

Save any changes that you made to the design object before closing the window.

Procedure

1. Select the object that needs to be salvaged.
2. Access the pulldown menu under **File > Salvage Object**.
3. A popup dialog box appears with a prompt asking if you want to try to save any pending changes.
4. Click the Yes button to save the changes.

Related Topics

[The Salvage Process](#)

Versions

Versioned design objects allow you to conveniently save previous states of your design.

In the Pyxis Project Manager, you can perform two types of version-related operations:

- Operations on the design object. These operations, which are available in the navigator, include changing the version depth of a design object and discarding the current version in favor of the previous one.
- Operations on a single version. These operations are available in the *version window*, which displays all versions of a design object, including the current version. Many of the operations that can be performed on design objects can also be performed on versions. For example, in the version window, you can copy, open, get information about, and delete a version.



Tip: For more information on design object versions, refer to the section “[Versions](#)” on page 37.

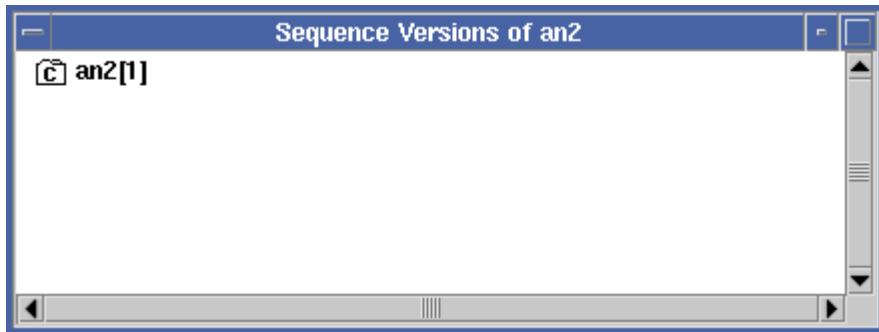
Showing the Versions of a Design Object	197
Getting Information about a Version	198
Opening a Version	199
Deleting from the Version Window	200
Deleting Multiple Back Versions	201
Copying a Version	202
Freezing and Unfreezing a Version	203
Showing Version Properties	203
Changing Version Depth	204
Reverting to a Previous Version	205

Showing the Versions of a Design Object

The Pyxis Project Manager shows the versions of a design object in the version window.

Figure 3-22, shows an example of the version window. In this example, *example_config* has three versions: 5, 6 and 7. The current version of a design object has the largest version number. In this example, version number 7 is the current version.

Figure 3-22. Version Window



Procedure

1. In a navigator, select the design object whose versions you want to view.
2. Choose **Report > Versions** from the popup or pulldown menu.
 - a. If the selected design object is a versioned object, the version window appears, displaying each version of the selected object. You can use the version window popup menu to operate on a selected version.
 - b. If the selected design object is not a versioned object, an error message appears stating that the object is not a versioned object. In this case, no version window appears.

Related Topics

[Getting Information about a Version](#)

Getting Information about a Version

Display information about a version.

Procedure

1. In the version window, select the version about which you want information.
2. Execute **Report Version Info** from the version window's popup menu. A read-only window appears, displaying the following information:
 - **Object Name.** The name of the design object whose versions are reported.
 - **Object Type.** The design object's type which defines the version's design object type.
 - **Location.** The version's absolute pathname, which is the pathname of this version of the design object.

- **Object Version.** The version number, which is an integer that is attached to each saved state (version) as the design object evolves. The first version of an object is always version number 1, the second version of an object is version number 2, and so on.
- **Version Frozen?** The version's state. A frozen version is not deleted by version pruning as the design object evolves.
- **Size.** The version's size.
- **Version Object Properties.** The properties of the design object of this version of the design object, if any exist. The properties are shown in the following format:

name: "value"

These properties are attached to the entire design object and accumulate as the object evolves.

- **Version Properties.** The version's properties, if any exist. The properties are shown in the following format:

name: "value"

These properties are attached to this particular version. Unlike object properties, you can view version properties only in this information window.

- **Version References.** The version's references, if any exist. The references are shown as the pathnames of the referenced objects. These references reflect the references that the design object held when this version was created. The current version of the design object may have quite a different set of references. When you select the whole design object in the navigator and show its references, you show the references held by the object's current version.

Related Topics

[Types](#)

[Properties](#)

[Freezing and Unfreezing a Version](#)

[References](#)

Opening a Version

In the version window, you can *open* a version, just as you can open a design object.

Opening a version invokes an appropriate tool for the design object version, either in read-only or edit mode, depending on the tool and which version you tried to open (some tools only open the current version). If you try to open a version that is not the design object's current version, you cannot edit that version.

Procedure

To open a version in the version window, perform one of the following steps:

- Double-click on the version's name or icon.
The default tool for the version's type invokes in read-only mode, with this version as input.
- Use the **Open** menu, according to the following steps:
 - a. Select the version, and execute **Open >** from the version window's popup menu.
The menu cascades to a list of tools that operate on this version's type.
 - b. Choose a tool from the list.
The next event depends on which version you selected and which tool you invoked:
 - i. If you open a version other than the highest numbered version, the tool does not allow you to edit the object. The tool invocation dialog box appears, only allowing you to invoke the tool in read-only mode; or the following error messages may appear:

```
Attempt to lock design object failed
Design object is a fixed version
```
 - ii. If you open the highest numbered version, the tool may invoke in edit mode. However, some tools, such as the Pyxis Project Manager configuration window, open all items in the version window in read-only mode.

Related Topics

[\\$\\$open_versioned_object\(\)](#)

Deleting from the Version Window

In the version window, you can delete one or more previous versions of a design object.

When you delete a previous version, the Pyxis Project Manager does not re-sequence the object's versions; the previous version is simply gone. In the version window, you cannot delete a frozen version.

Procedure

1. If you have not already displayed the version window, do so by selecting the design object in the navigator and choosing the **Report > Show Versions** pulldown or popup menu item.
2. In the version window, select the version that you want to delete.
3. Choose **Delete** from the version window popup menu.

A dialog box displays asking you if you want to delete the selected versions.

4. Click the **Yes** button.

The selected version is deleted, and the version window updates its display.

Related Topics

[Freezing and Unfreezing a Version](#)

Deleting Multiple Back Versions

Remove old versions of a design object.

Procedure

1. Select the design object in the navigator window.
2. Choose the following pulldown or popup menu item: **Edit > Delete > Excess Versions**.
3. In the resulting prompt bar, specify the number of versions you want to keep, in the **keep** field.
4. Specify whether you want the delete operation to include all levels of design hierarchy beneath the selected object (containment) or whether you want to delete only the object itself (object).
5. In the **mode** field, specify:

Whether you want the delete operation to include all levels of design hierarchy beneath the selected object (containment),

or

Whether you want to delete only the object itself (object).

or

Whether you want only the references to the versions deleted (reference).

or

Whether you want to not select any of the above modes (void).

6. Confirm the deletion by clicking the **OK** button.

The selected design object's versions are permanently deleted, based upon the mode selected.

7. OK the prompt bar.

Copying a Version

In the version window, you can select and copy a previous version of a design object.

At the destination of your copy, the copied version becomes version number 1 of a complete new design object. If the version is a container design object, the Pyxis Project Manager updates references within the container to reflect the new location during the copy operation.

Procedure

1. In the version window, select the (one) version that you want to copy.
2. Execute **Copy** from the version window's popup menu, which displays a prompt bar.
3. Press the prompt bar's **Options** button to display the Copy Version Options dialog box.

This dialog box allows you to specify how the Copy Version command overwrites design objects of the same name:

- **Cancel** the entire copy operation if any name conflicts are found.
- **Replace** conflicting design objects with copies of the selected design objects.
- **Ask** if you want to leave the destination object intact, or replace it with a copy. This is the default.

4. Execute the dialog box, which returns you to the prompt bar.
5. In the prompt bar's Destination field, enter the directory to which you want to copy the selected objects:
 - If the directory exists, the selected version is copied into that directory.
 - If the directory does not exist, the Copy Version command copies the selected version to that name, whether the selected version is a directory or not. However, this only occurs if everything in the destination pathname exists except the leaf; that is, this function does not create intermediate directories.

Instead of specifying the destination as a full pathname, you can specify the destination as only a *leaf name*. A leaf name is the portion of a pathname following the last “/”. If you specify a leaf, the copied version is added in the current directory of the navigator from which you displayed versions. For example, to copy a version of design object *tom* to the name *phil* in the same directory, simply specify “*phil*” in the Destination field.

6. Execute the Copy Version prompt bar by pressing the Return key.

Related Topics

[\\$copy_version\(\)](#)

Freezing and Unfreezing a Version

As you evolve a design object, you create new versions, and those versions that exceed the object's version depth are *pruned* (deleted) automatically.

In the version window, you can prevent versions from being pruned by freezing them.

While a version is frozen, the following conditions are in effect:

- You cannot delete the version in the version window.
- The version cannot be pruned as the design object evolves.
- You cannot delete the version by reverting to the previous version in the navigator.

Later, you can unfreeze the frozen version. If the newly unfrozen version is outside of the design object's version depth, when you create a new version, the unfrozen version is deleted. Freezing a version does not prevent you from deleting the entire design object, including the frozen version, while in the navigator. You can determine whether a version is frozen by viewing the Report Version Info window.

When you freeze a version, you actually add a file to the design object's fileset, called a *freeze file*.

Procedure

1. In the version window, select the version that you want to freeze or to unfreeze.
2. To freeze the selected version, choose **Freeze** from the version window's popup menu.
3. To unfreeze the selected version, choose **Unfreeze** from the version window's popup menu.

Related Topics

[Design Object Filesets](#)

Showing Version Properties

You can execute the version window's popup menu item **Report Version Info** to display the properties of a particular version.

This menu item reports all of the known information about the selected version, including its versions properties.

A version property has two values associated with it: the property's name and the property's value. You can change either of these values from the navigator window using the Navigator's pulldown or popup menu item **Edit > Change > Property**.

Related Topics

[Getting Information about a Version](#)

Changing Version Depth

A design object's version depth specifies the number of previous versions that are saved on disk.

The current version of a design object is included in the total version count. As you evolve a design object and create new versions, those versions that are outside the version depth are deleted automatically.

Every versioned design object has a default version depth, which is specified by the object's type. In the Pyxis Project Manager navigator, you can change the version depth so that your design object saves more or fewer previous states. If you decrease a design object's version depth, those versions outside the new version depth that are not frozen or labeled are immediately deleted. The deletion of previous versions is referred to as *pruning*.

You can determine a design object's current version depth by viewing the Report Object Info window.

Procedure

1. In a navigator, select the design object whose version depth you want to change.
2. Choose **Edit > Change > Version Depth** from the navigator's popup or pulldown menu, which displays a dialog box asking you if you want to proceed with the task, even though reducing the version depth may delete back versions.
 - a. If you click the Yes button, it brings up the CHAnge VErsion Depth prompt bar.
 - b. If you click the No button, the task is not executed further.
3. In the New Depth field, enter an integer.
For an infinite version depth, enter “-1”.
4. In the Mode field, select the mode by clicking on the up and down arrow buttons.
 - If you select the Object option, the version depth of the selected design object is changed. The version depth of the design objects that are either referenced by or contained within the selected design object are not changed.
 - If you select the Containment option, the version depth of the selected design object and of all of the objects contained within the selected design object is changed.
 - If you select the Reference option, the version depth of the selected design object and of all of the objects contained within or referenced by either the selected object or by any objects contained within the selected design object is changed.

5. Execute the prompt bar by clicking **OK**.

The version depth of the specified design objects is changed and the versions that exceed the version depth are immediately pruned off.

Related Topics

[Getting Information about a Design Object](#) `$change_version_depth()`
[\\$\\$prune_design_hierarchy\(\)](#)

Reverting to a Previous Version

One of the most common version management tasks is to discard the current version of a versioned design object so that you can use the previous version.

This process is known as *reverting* to the previous version. When you revert the design object, the previous version becomes the new current version.

You can revert to the previous version by deleting the current version in the version window, but the Pyxis Project Manager also allows you to do this in the navigator. If the current version is frozen, you cannot delete it with the Revert Version command.

Procedure

1. In a navigator, select the versioned design object whose previous version you want to make current.

The selected design object must be a *versioned* design object.
2. Choose **Edit > Revert Version** from the navigator's pulldown menu, which displays a dialog box asking if you want to continue.
3. Click the **Yes** button, which deletes the current version and makes the previous version become the current version.

Related Topics

[\\$revert_version\(\)](#)

References

A reference is a pointer from one design object to another. Applications create references to associate design objects, and you can create references in the Pyxis Project Manager.

In addition to a pathname, references point to a particular version of the design object. References can also hold properties.

References are also unidirectional. When exploring a reference, remember that the parent of the reference is the object from which you explored the reference, not the parent as defined in the pathname specified by the reference.

In the Pyxis Project Manager, you can perform two types of reference operations:

- Operations on the Design Object

These operations, which are available in the navigator, include adding a reference to a design object, displaying an object's references, and changing the reference pathnames of all references that are held by a design object.

- Operations on the Reference

These operations are available in the *reference window* which displays a design object's references. They include getting information about an individual reference, deleting a reference, and editing a property to a reference, such as showing, adding, changing, and deleting a property.



Tip: For more information about references, refer to the section “[References](#)” on page 39.

Showing the References of a Design Object	206
Getting Information about a Reference	207
Add and Delete a Reference	208
Change References	210
Convert Hard References to Soft References	214
Checking for Broken References	217
Fixing Reference Problems Due to Viewpoints and Part Interfaces	220
Working with Reference Properties	220

Showing the References of a Design Object

You view a selected design object's references in the reference window. Once there, you can get information about and operate on a single reference.

Procedure

1. In a navigator, select the design object whose references you want to view.
2. Choose **Report > Show References > For Object** from the navigator's popup or pulldown menu.

A reference window, similar to the one in Figure 3-23, appears. This window lists all of the references held by the selected design object. If the selected design object does not have any references, an empty window appears.

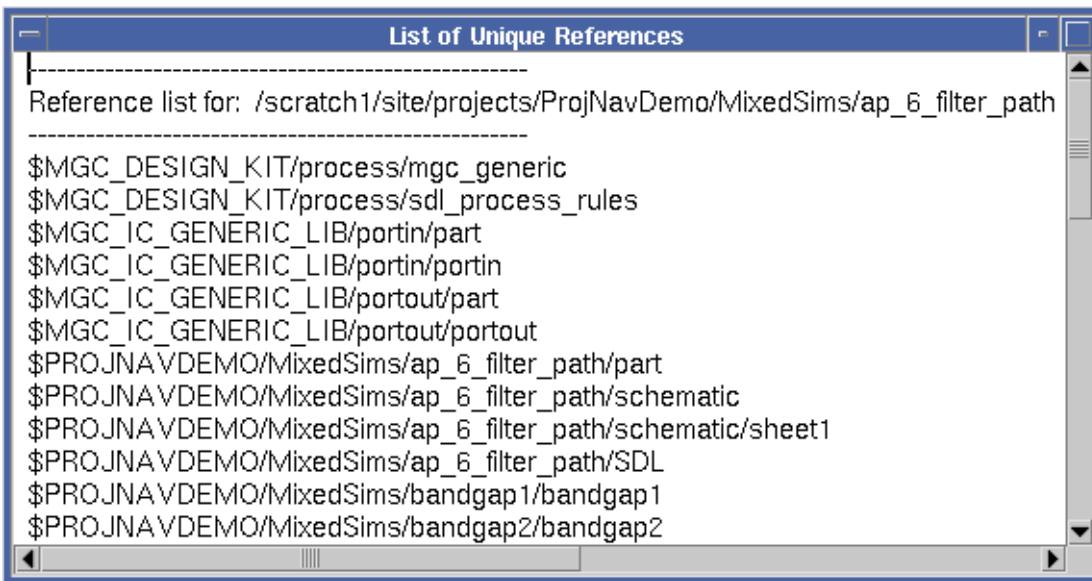
Each displayed reference shows the referenced object's type, as indicated by the small icon and pathname.

The reference window displays exactly the same items as the navigator does when you select a design object and explore its references. Do not confuse these two windows. The navigator displays design objects, and the reference window displays references. Deleting an item in the navigator deletes a design object. Deleting an item in the reference window deletes a reference (a pointer) to a design object.

3. Choose **Report > Design References** from the navigator's popup or pulldown menu.

A reference window called “List of Unique References” appears.

Figure 3-23. Example of a Reference Window



Related Topics

[\\$show_references\(\)](#)

Getting Information about a Reference

In the reference window, you can get information about a reference's characteristics and current status.

Procedure

1. In a reference window, select the reference about which you want information.
2. Execute **Report Reference Info** from the reference window's popup or pulldown menu.

A read-only window appears, displaying the following information:

- **Object Name.** The name of the design object that holds the reference.
- **Object Type.** The type of the design object that holds the reference.
- **Location.** The absolute pathname of the design object that holds the reference.
- **Reference Path.** The absolute pathname of the referenced design object.
- **Reference Type.** The type of the referenced design object.
- **Referenced Version Number.** The version number of the referenced design object to which this reference pointed when the reference was last saved. If this reference points to a fixed version, this number may not be the referenced design object's current version number.
- **Reference Path Type.** The referenced design object's reference path type, either soft or hard:
 - A hard reference pathname begins with a “/”.
 - A soft reference pathname begins with a “\$”.
- **Reference State.** The referenced design object's state. Every reference can have one of the following states:
 - Current. The reference points to the current version of the referenced object. As that object evolves, the reference still always points to the most current version.
 - Fixed. The reference points to a particular version of the referenced object. As that object evolves, this reference still points to the same version.
 - Read-only. An application that accesses the referenced object through this reference can read but not edit the object.
- **Reference Properties.** The reference's properties, if any exist. The properties are shown in the following format:

name: “value”

Related Topics

[Design Management with Location Maps](#) [Properties](#)

Add and Delete a Reference

When you reference a design object using a relative pathname that does not begin with a “\$”, the relative pathname is converted to an absolute pathname based on the value of the environment variable MGC_WD.

You must ensure that the value of MGC_WD is set to the correct value for your current working directory or an incorrect pathname for the reference may be stored.

Adding a Reference.	209
Deleting a Reference.	209

Adding a Reference

You can add references to a design object in the Navigator or Project Navigator window, and later display the reference by using the **Report > Show References** pulldown menu item.

Procedure

1. Select the design object for which the reference needs to be added.
2. From the Navigator's or Project Navigator's pulldown menu, choose the **File > Add Reference** option.

or

From the Navigator's popup menu, choose **Add > Reference**.

3. Both the above options bring up the Select Reference Destination **dialog box**.
4. Navigate to the directory location of the reference.
5. Type the filename for the reference in the File name field.
6. Click the OK button to execute the Select Reference Destination **dialog box**.

The Pyxis Project Manager creates a reference from the object that you selected to the object that you chose in the object browser dialog.

Related Topics

[\\$add_reference\(\)](#)

Deleting a Reference

The Pyxis Project Manager can delete only the references that it has created itself.

References that the Pyxis Project Manager creates always hold the “creating_tool” “design manager” property. This property tells the Pyxis Project Manager that it can delete that reference; it cannot delete a reference that does not hold this property. You can delete a reference only from the reference window.

Procedure

1. Display a reference window, by executing **Report > Show References** from the navigator's popup or pulldown menu.
2. Select the reference that you want to delete.

3. Choose **Delete** from the reference window's popup menu.

A dialog box appears, asking you to confirm the deletion.

4. Click **Yes**.

The reference window updates and the selected reference disappears from the reference window.

Related Topics

[\\$delete_reference\(\)](#)

Change References

The references of a design object can be changed. You can modify a reference pathname or state.

Changing Reference Pathnames	210
Changing a Reference's State	213

Changing Reference Pathnames

Moving and deleting design objects break the references that point to the moved or deleted objects.

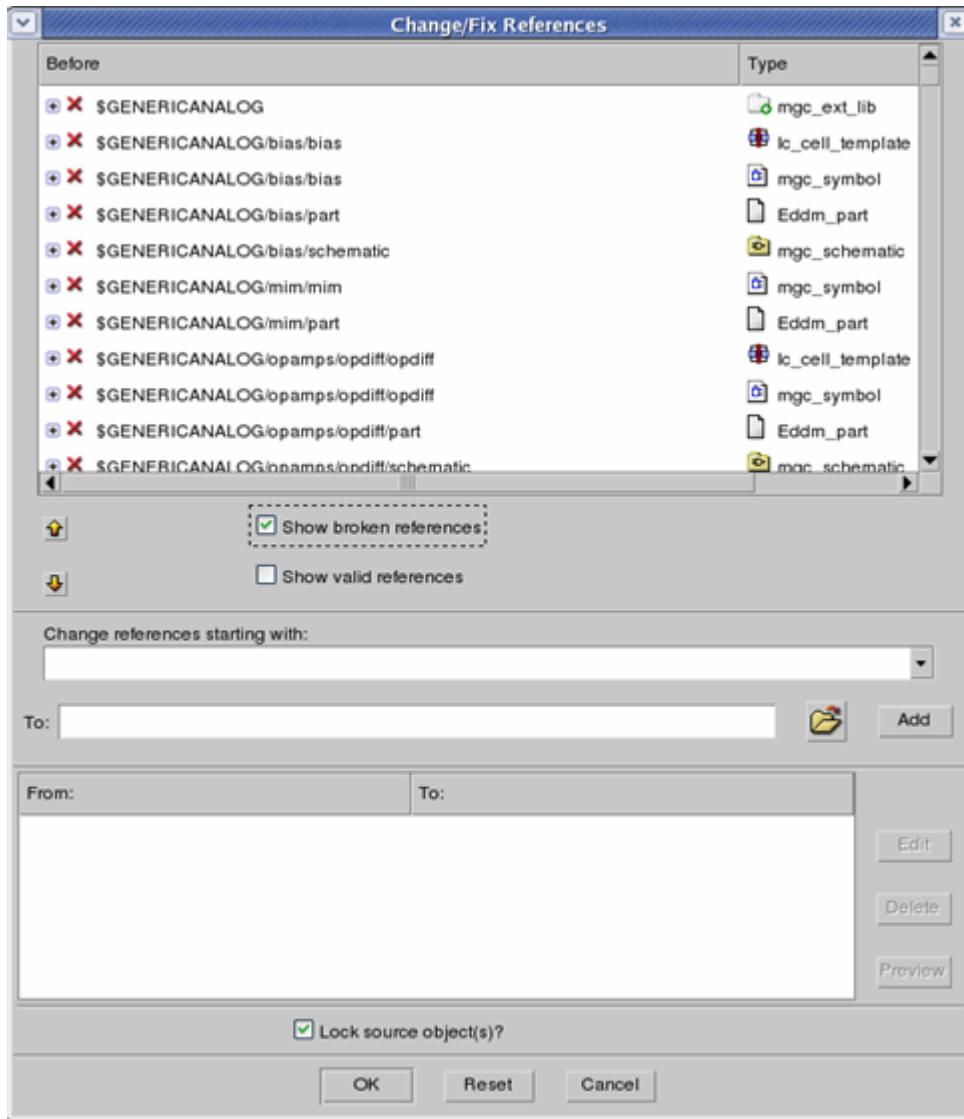
For this reason, after you move or delete design objects, you sometimes need to change the reference pathnames of other design objects. For this operation, the Pyxis Project Manager provides the Change Object References command.

You can change the references of a large group of objects simultaneously by including them in a configuration and using the Change Configuration References command.

Procedure

1. In a navigator, select the design objects whose references you want to change.
2. Choose **Edit > Change > References** from the Navigator or Project Navigator window's popup or pulldown menu. The Change/ Fix References dialog box appears, as shown in Figure 3-24.

Figure 3-24. Change/Fix References Dialog Box



3. The main options have the following meaning:

- **Show broken references**

If you check this option, then the list of references in the top (Before) pane includes all the broken references (for the selected design object) whose pathnames need to be changed or fixed. By default, this option is selected.

- **Show valid references**

If you check this option, then the list of references in the top (Before) pane includes all valid references for the specified design object.

- **Change references starting with**

Enter the portion of the reference pathname that you want to change.

You can specify this searched-for pattern using UNIX System V regular expressions (except for the wild card character, the asterisk “*”). If you specify a pattern that exists in more than one portion of a reference, only the first occurrence of the pattern is replaced. The following list shows an example of this behavior:

Original reference	usr2/jonb/designs/jonb_design3
From field	onb
To field	redh
Resulting reference	usr2/fredh/designs/jonb_design3

- **To**

In the To field, enter the text pattern that you want Pyxis Project Manager to substitute. You should enter soft pathnames whenever possible.

As you enter literal text or a regular expression into the From field, another set of From and To fields appear. After you have entered all your substitutions, use the mouse to move the cursor to the remaining fields on the dialog box.

4. The additional fields allow you to specify whether you want to change the references of the current version or all versions of the selected design objects, to specify if you want to preview the execution of the command or execute the command, and to specify whether you want to lock the source objects during the operation.

By default, the **Edit > Change > References** menu item changes the references of the current version, executes the command, and locks the source design objects.

5. Select the option you want by clicking on the desired button.
 - If you click on the Preview button, the command executes.
 - If you click on the Edit button, you can change the options.
 - If you click on the Delete button, the selected option is deleted from the list.
6. In the Lock Source Object(s) field, select this option you want by clicking on the button.
 - If you check this field, the source design objects are locked before their references are changed.
 - If you do not check this field, the source design objects are not locked before their references are changed.
7. Click **OK**, to execute the Change/ Fix References prompt bar.

Pyxis Project Manager searches the references of each selected design object. When it finds a reference whose pathname contains the From Pattern, it substitutes the To Pattern.

The **Edit > Change > References** menu item allows you to change the references of the current version or of all versions of each selected design object. If you change the references of just the current version of the selected design object, the previous versions of each object still hold the old references. As a result, when you revert to a previous version, you may have to change references again.

The **Edit > Change > References** menu item is the menu interface to the Change/ Fix References command. You can interrupt the Change/ Fix References command, at any time, by pressing the Ctrl and Backlash keys together.

When you interrupt the operation in this way, a dialog box is displayed prompting you about whether the operation should halt or continue. If you choose to halt the operation, the operation is aborted and the standard clean-up procedure is performed.

Both the Change/ Fix References and Change Configuration References commands provide a @preview switch that allows you to transcript the results of their simulated execution, without actually making any changes to the references.

Related Topics

[Changing the References of a Design Configuration](#)

`$change_object_references()`

[Pathnames in the Pyxis Project Manager](#)

`$change_configuration_references()`

Changing a Reference's State

References have states, and you can change the state of references that were added by users in the Pyxis Project Manager.

The following states are valid reference states:

- **Current.** A current reference always points to the current version of the referenced object. When you create a new version of the referenced object, a reference with this state always points to the newly created current version. This state is the most common and is the default when you add a reference in the Pyxis Project Manager.
- **Read-only.** A read-only reference points to the current version of a design object, but you cannot edit this object when you access it through the reference, regardless of the access permissions of the referenced object.
- **Fixed.** A fixed reference points to a particular version of a design object. Fixed references are always read-only references. You can only fix a reference while pointing to the current version of an object.

Procedure

1. In a navigator, select the design object whose reference state you want to change.

2. Choose **Report > Show References** from the navigator's popup or pulldown menu, which displays a reference window for the selected object.
3. In the reference window, select the reference whose state you want to change.
4. Choose **Change State** from the reference window's popup menu, which displays a prompt bar.
5. Select the state that you want by clicking on the up or down arrow key in the Reference State field. You may select one of the three possible reference states: current, fixed, or read-only.
6. Click **OK**. The state of the selected references changes to your choice.

Related Topics

[\\$change_reference_state\(\)](#)

Convert Hard References to Soft References

If you began your design before your site was using location maps or the location map at your site is changed to include additional soft prefixes, you should convert the references on your design from hard references to soft references.

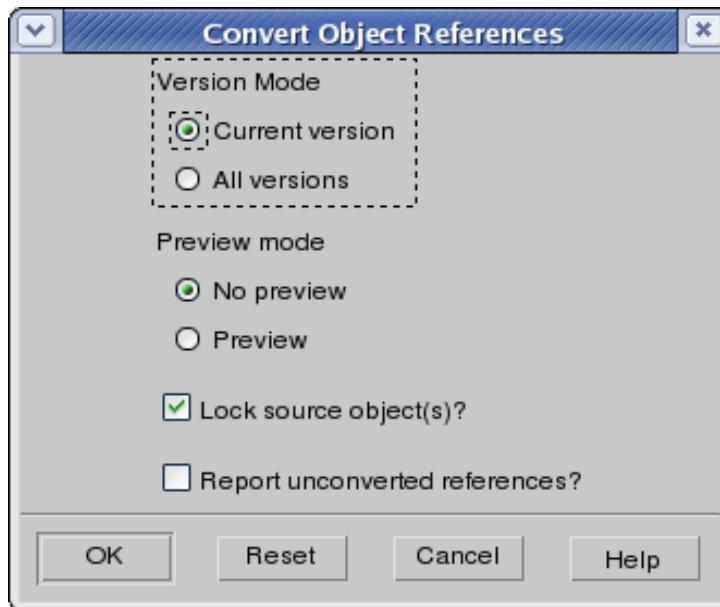
Converting Design Object References	214
Converting Configuration Object References	216

Converting Design Object References

Convert hard references to soft references on a design object or a set of design objects.

Procedure

1. In a navigator, select the design objects whose references you want to convert.
2. Execute the pulldown menu item **Edit > Convert References**, which brings up the Convert Object References dialog box, as shown in [Figure 3-25](#).

Figure 3-25. Convert Object References Dialog Box

3. In the Version Mode field, select the option you want by clicking on the desired button.
 - If you click on the **Current Version** button, the references of the current version of the selected design objects are converted.
 - If you click on the **All Versions** button, the references of all versions of the selected design objects are converted.
4. In the Preview Mode field, select the option you want by clicking on the desired button.
 - If you click on the **No Preview** button, the command executes.
 - If you click on the **Preview** button, the results of the command's execution are reported to the monitor window, but the command does not actually execute.
5. In the Lock Source Object(s) checkbox, select the option you want by clicking it.
 - If you click this option, the source design objects are locked before their references are converted.
 - If you do not click this checkbox, the source design objects are not locked before their references are converted.
6. In the Report Unconverted References field, select the option you want by clicking on the checkbox.
 - If you click this option, all reference pathnames that cannot be converted to a soft pathname based on the entries in the current location map are reported to the monitor window.

- If you do not click this checkbox, no information is reported about reference pathnames that cannot be successfully converted to soft pathnames.
7. Click **OK**, to execute the Convert Object References dialog box.
- The Pyxis Project Manager searches the references of each selected object. When it finds a hard reference that has a matching entry in the current location map, it substitutes the soft reference for the hard reference.

Related Topics

[\\$convert_object_references\(\)](#)

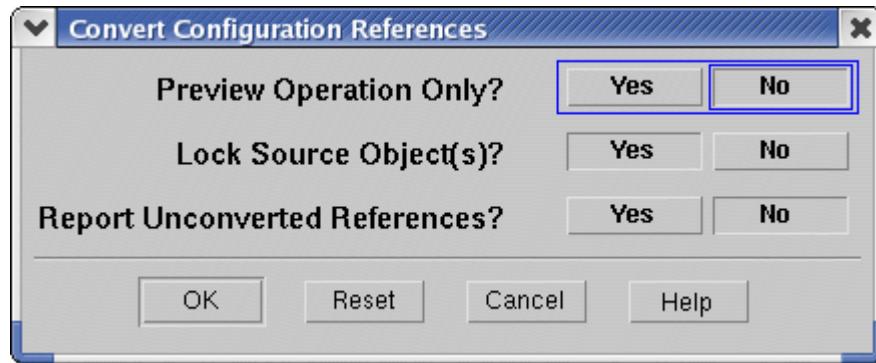
Converting Configuration Object References

Convert hard references to soft references in a configuration object.

Procedure

1. Open the configuration.
2. Execute the pulldown menu item **Edit > Convert References**, which brings up the Convert Configuration References dialog box, as shown in [Figure 3-26](#).

Figure 3-26. Convert Configuration References Dialog Box



3. In the Preview Operation Only field, select the option you want.
 - If you click on the **No** button, the command executes.
 - If you click on the **Yes** button, the results of the command's execution are reported to the monitor window, but the command does not execute.
4. In the Lock Source Object(s) field, select the option you want by clicking on the desired button.
 - If you click on the **Yes** button, the source design objects are locked before their references are converted.

- If you click on the **No** button, the source design objects are not locked before their references are converted.
5. In the Report Unconverted References field, select the option you want by clicking on the desired button.
- If you click on the **Yes** button, all reference pathnames that cannot be converted to a soft pathname based on the entries in the current location map are reported to the monitor window.
 - If you click on the **No** button, no information is reported about reference pathnames that cannot be successfully converted to soft pathnames.
6. Click **OK**, to execute the Convert Configuration References dialog box.

The Pyxis Project Manager searches the references of each item in the configuration object. When it finds a hard reference that has a matching entry in the current location map, it substitutes the soft reference for the hard reference.

Related Topics

[\\$convert_configuration_references\(\)](#)

Checking for Broken References

Moving and copying design objects can break the references that point to moved or deleted design objects.

For this reason, you should always check your design objects before and after you have performed a move or copy operation on them. Pyxis Project Manager provides the Check References command for this purpose.

Checking References Using the Command Prompt	217
Checking References Using the Pulldown Menu	219

Checking References Using the Command Prompt

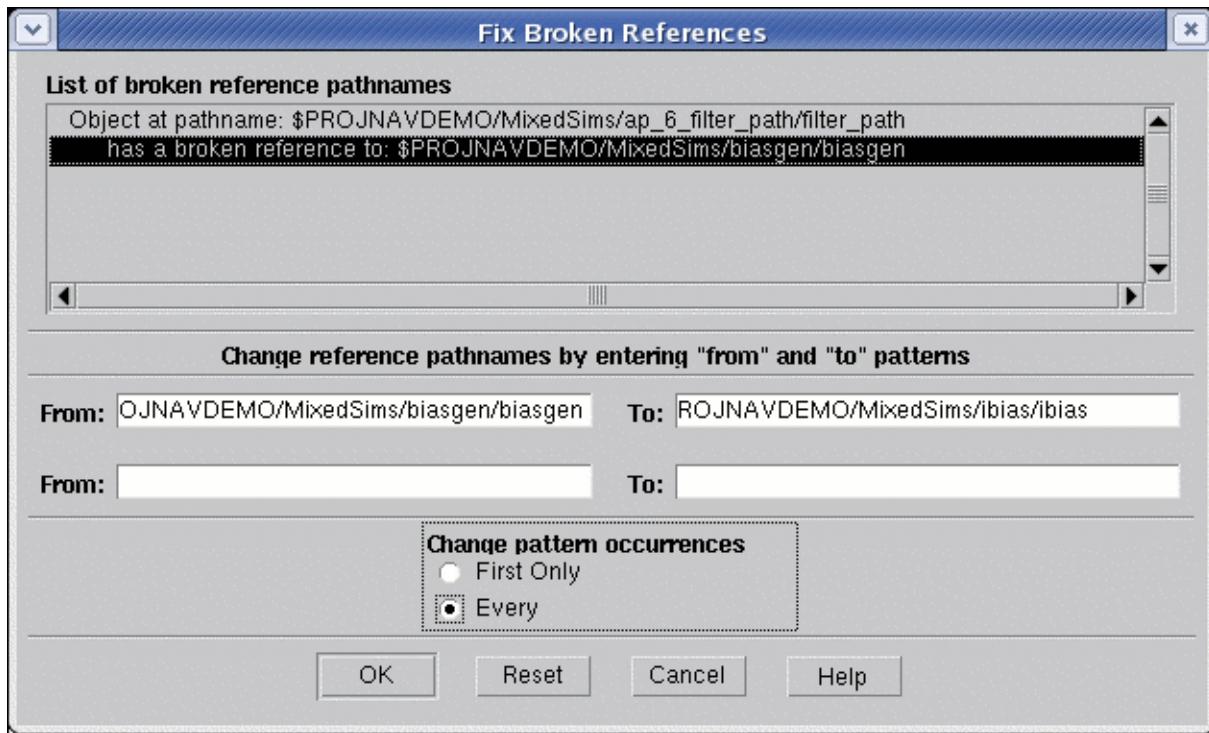
Check references of a design object using the command prompt.

Procedure

1. Select the design object whose references need to be checked.
2. In the command prompt, type the CHEck REferences command. Alternately, type \$check_references() at the prompt.
 - a. If there are no broken references, the message is displayed accordingly in the Message Area window.

- b. If there are broken references, it opens up the Fix Broken References dialog box.
3. The Fix Broken References dialog box, as shown in [Figure 3-27](#) displays a scrollable list of all broken references and an expanding Change Reference Pathnames field. You can use the Change Reference Pathnames field to fix your broken references.

Figure 3-27. Fix Broken References Dialog Box



4. To “fix,” or redirect the broken references that you see reported in the Fix Broken References dialog box, perform the following steps:
 - a. In the From: field, enter the string portion of the broken reference pathname that you want to replace.
 - b. In the To: field, enter the correct string that you want substituted in place of the corresponding From: field string.
 - c. Repeat steps 4 a.) and 4 b.) until you have specified all of the string replacements that you want to make.

All of the references that match the From: field pattern are changed to the To: field pattern. Although you can scroll through the list area to locate those strings that you want to replace, selecting a reference in the list area has no effect.

5. Execute the Fix Broken References dialog box by clicking the OK button.

The string replacements you specified are performed. A dialog box, containing the following question, displays allowing you to specify whether you want to verify that your changes have successfully repaired the references.

References were changed. Do you want to check for more broken references?

You should click on the **Yes** button to verify that your references are repaired.

6. Repeat step 5 until the following message appears in the message area verifying that your references are fixed.

No broken references were found.

Related Topics

[\\$check_references\(\)](#)

Checking References Using the Pulldown Menu

Check references of a design object using the pulldown menu.

Procedure

1. Select the design object whose references need to be checked.
2. From the pulldown menu, choose the **Edit > Check References** option, which brings up the Change/Fix References **dialog box**, as shown in Figure 3-24.
3. The references listed in the Change/Fix References **dialog box** are preceded with either of the following symbols to indicate whether they are broken or valid, as shown in Table 3-2.

Table 3-2. Symbol meaning

Symbol	Meaning
	Broken reference
	Valid reference

The Change References section in this chapter explains the options in the Change/Fix references dialog box.

4. The Check References command transmits its progress to the Message Area window. When broken references are reported, the Change/Fix References dialog box, as shown

in Figure 3-24, displays a scrollable list of all broken references. You can also use the expanding “Change references starting with” field to fix your broken references.

Related Topics

[\\$check_references\(\)](#)

Fixing Reference Problems Due to Viewpoints and Part Interfaces

Many times when you find broken references, the problem is caused by faults with the viewpoint or a part interface.

To fix these problems perform the following actions. If the problem is caused by viewpoints, skip step 1 and proceed to step 3.

Procedure

1. Invoke:

```
$MGC_HOME/bin/cib_ic/path_to_top_of_design
```

2. Perform the following cib commands:

- a. View the design and obtain model number of old schematic.
- b. Invoke the command: unregister_model old_model_number.
- c. Save and Quit out of cib.

3. Invoke DVE and perform the following menu command:

```
File > Save Viewpoint > With Same Name > Cleanup Un-used References
```

4. Save and Quit out of DVE.

5. Return to Pyxis Project Manager and check the references again.

Related Topics

[\\$check_references\(\)](#)

Working with Reference Properties

In the Pyxis Project Manager you can add, change, or delete a reference's properties.

Adding, Changing, and Deleting Reference Properties	221
Showing Reference Properties	221

Adding, Changing, and Deleting Reference Properties

In the reference window, you can add, change, and delete properties on references. All three operations work similarly.

Procedure

1. In the reference window, select the reference to which you want to add, change, or delete a property.
2. Choose one of the following items from the reference window's popup or pulldown menu:
 - a. To add a property, choose Reference Properties > Add.
 - b. To change an existing property, choose Reference Properties > Change.
 - c. To delete a property, choose Reference Properties > Delete.
3. Enter the property that you want to add, change, or delete.
4. Click OK.

Related Topics

[\\$add_reference\(\)](#)

[\\$change_reference_property\(\)](#)

[\\$delete_reference_property\(\)](#)

Showing Reference Properties

You can see the results of adding, changing, or deleting a reference property by selecting the reference in the reference window and displaying its properties.

Procedure

1. In a reference window, select a reference whose properties you want to view.
2. Choose Report Reference Info from the popup or pulldown menu.
A Reference Information Report window appears, displaying information about the reference, including a list of its properties.

Related Topics

[\\$show_references\(\)](#)

Design Object Properties

You can add properties to design objects and references.

Once a property exists, you can change or delete it. The basic approach you use when adding a property is to first select the design object and then add the property. You can change and delete properties the same way.

Showing the Properties of a Design Object	222
Adding an Object Property	222
Deleting a Property	223
Changing an Object Property	223
Changing Reference Properties.	224

Showing the Properties of a Design Object

You can view a selected design object's properties in a read-only Object Information Report window.

Procedure

1. In a navigator, select the design object whose properties you want to view.
2. Choose **Report > Object Info** from the navigator's popup menu.

Related Topics

[\\$report_object_info\(\)](#)

Adding an Object Property

Add properties to design objects using the pulldown menu, from the Navigator or Project Navigator window.

Procedure

1. Select the object to which you want to add the property.
2. Choose **File > Add Property** from the pulldown menu, which displays the ADD OBject Property prompt bar.
3. Enter the property name that you want to add in the Prop Name field.
4. Enter the property value that you want to add in the Prop Value field.
5. Confirm the addition of the property by clicking the **OK** button. The new property is added to the selected design object.

Related Topics

[\\$add_object_property\(\)](#)

Deleting a Property

The deletion of a design object's property is performed by executing the Delete Object Property command, which can be accessed through the pulldown or popup menu, or by typing the command in the popup command line.

You can delete a property from a Navigator or Project Navigator window, and later verify the deletion by using the navigator's popup menu item **Report Object Info**.

Procedure

1. Select the object whose property you want to delete.
2. Choose **Edit > Delete > Property** from the popup or pulldown menu, which displays the DElete OBject Property prompt bar.
3. **At the prompt, type in the name of the property that you want deleted.**
4. Confirm the deletion of the property by clicking the **OK** button. The selected design object property is permanently deleted. You cannot retrieve it.

Related Topics

[\\$delete_object_property\(\)](#)

Changing an Object Property

Change a property for a design object.

Procedure

1. Select the object whose property you want to change.
2. Choose **Edit > Change > Property** from the pulldown menu, which displays the CHAnge OBject Property prompt bar.
3. Enter the name of the property you want to change, in the Prop Name field.
4. After you enter the property name, you can change the property value, in the New Value field. If you enter the property name incorrectly, no change occurs.
5. Confirm the addition of the property by clicking the **OK** button. The specified property name and/or value are changed for the selected design object.

Related Topics

[\\$change_object_property\(\)](#)

Changing Reference Properties

You can add, delete, change, and show properties on references.

You can use reference properties to store various kinds of information. For example, you may wish to record when the reference was added, who added it, or how the two objects that are connected by the reference are related.

Related Topics

[Working with Reference Properties](#)

Change Your Session Setup

You can modify specific session characteristics by using the pulldown menu item **Setup** > which is available from all Pyxis Project Manager windows.

After modifying these session characteristics, you can specify whether the Pyxis Project Manager automatically saves these settings in your Pyxis Project Manager default startup file when you end your session. If you specify to save these settings in your default startup file and the default startup file is the only startup file executed, the next time that you invoke the Pyxis Project Manager the setup values you specified in your last Pyxis Project Manager session are used.

Save Session Setup Settings	224
Specifying the Default File Editor	225
Changing the Appearance of Iconic Windows	226
Specifying Startup Windows	228
Specifying Session Defaults	231
Specifying Navigator Filters	232
Specify Configuration Monitoring	236

Save Session Setup Settings

The Pyxis Project Manager provides you with the ability to modify individual session setup characteristics and to save them in your Pyxis Project Manager default startup file.

The first time that you change a specific session setting, using any one of the menu items available from the pulldown menu item **Setup**, the Pyxis Project Manager displays a dialog box that asks you if you want to save the new settings in your default startup file. If you choose to save the new settings, the current changes and all subsequent changes you make in that session are saved. If you choose not to save the new settings, the current changes and any subsequent

changes you make in that session are discarded when you exit the Pyxis Project Manager. If you choose not to save the first setup change and you decide later in the session that you want to save the setup changes that you have made, you can override your first decision by executing the pulldown menu item **Setup > Write Default Startup File**.

Note

 When you run a ‘here’ document, by invoking the Pyxis Project Manager with the `-nodisplay` switch, and change either the toolbox search path or a setup characteristic, the Pyxis Project Manager prompts you whether to save the changes to a startup file. Because you are running in server mode, neither of these prompts are visible and the Pyxis Project Manager appears to hang while waiting for input. To prevent this from occurring, in your “here” document you need to set the value of the external variable `$MGC_SAVE_SETUP` to `@no` and the value of the external variable `$toolbox_search_path_changed` to `@false`.

The Pyxis Project Manager always saves your changes in the Pyxis Project Manager default startup file `dmgr_default.startup`. However, when you specify to save your changes, the Pyxis Project Manager first checks for the existence of any other startup files. The Pyxis Project Manager performs this check because all other startup files take precedence over the default startup file `dmgr_default.startup` at invocation. If other startup files exist, the Pyxis Project Manager prompts you that a site, project, or user-specific startup file exists and asks if you still want to save your settings in the default startup file. If you choose to save your settings in the default startup file and another startup file exists, the new settings are saved in your default startup file, but they are not used when you invoke the Pyxis Project Manager.

Because the **Setup > Navigator Filters > Active Navigator** pulldown menu item specifies filters for the active navigator only, you are never prompted about whether you want to save your filter settings. When you execute the **Setup > Navigator Filters > Active Navigator** pulldown menu item, your new filters settings are applied to the active navigator only and are not saved in the default startup file.

Related Topics

[dmgr_ic](#)

Specifying the Default File Editor

When you open a file in the Pyxis Project Manager, a default file editor opens the file.

This same file editor invokes when you double-click the *Editor* tool in the tools window. Unless you change the default editor setting by using the **Setup > Preferences > Session tab**, the Pyxis Project Manager default editor is the Notepad editor.

Procedure

1. In any window, choose **Setup > Preferences > Session** from the pulldown menu, which displays a dialog box with a Default Text Editor. You can choose either Notepad or the User-specified editor as your default editor.
2. The dialog box options have the following meaning:
 - Notepad (MGC Internal Notepad Editor) option: if you choose Notepad, executing the **File > Open** menu item on an ASCII file causes the Notepad editor to invoke on that file. The Notepad editor is the default editor.
 - Editor path
If you selected the User-specified option, then use this field to select the editor of your choice by typing in or browsing to the editor path. This option is used when an ASCII file needs to be opened in read/write mode in Pyxis Project Manager.
 - Read-only editor path
If you selected the User-specified option, then use this field to select the editor of your choice by typing in or browsing to the editor path. This option is used when an ASCII file needs to be viewed in Pyxis Project Manager.

For example, to specify the **vi** editor as the read/write editor, you would enter **/usr/bin/vi** at the Editor pathname field. To specify **view** as the read-only editor, you would enter **/usr/bin/view** in the Read-only Editor pathname field.

If either the read/write editor or the read-only editor pathnames do not exist, an error message is displayed when you attempt to execute the dialog box. Execute the Default Editor Setup dialog box by clicking **OK**.

The next time that you open a file, the specified editor is invoked.

If this is the first change you have made to your setup characteristics in the current session, you are prompted whether to save the new setup values to your default file. If you choose to save the setup characteristics that you specified in the Default Editor Setup dialog box, the default editor setting is saved in the Pyxis Project Manager default startup file *dmgr_default.startup*. As a result, if no other startup files exist, the next time that you invoke the Pyxis Project Manager, it uses your newly specified default editor.

Related Topics

[Change Your Session Setup](#)

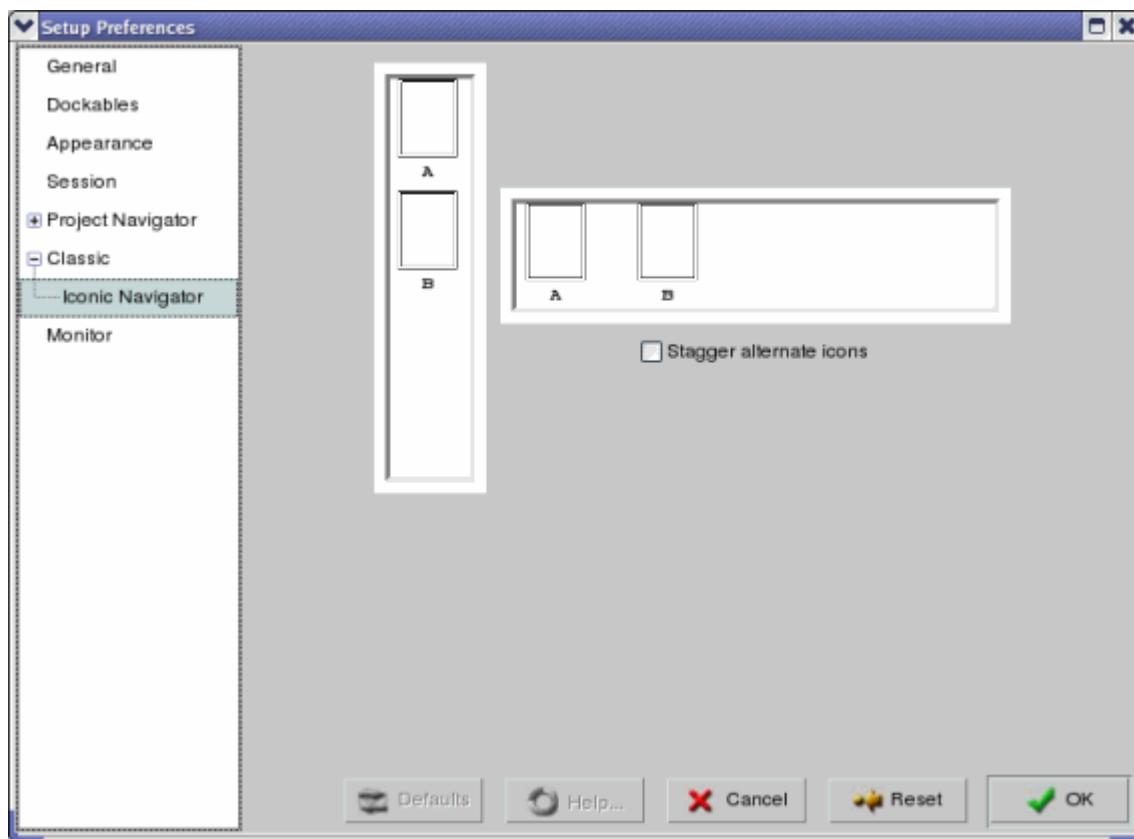
Changing the Appearance of Iconic Windows

By default, the Pyxis Project Manager displays the icons in the iconic navigator and tools window next to one another.

The Pyxis Project Manager calculates the number of columns needed to display the icons based on the width of the window, so that all icons are visible by scrolling vertically in the window.

The spacing between each icon is provided as a default. When you execute the **Setup > Preferences > Classic tab** pulldown menu item, the Iconic Navigator tabbed dialog, as shown in Figure 3-28, allows you to change the horizontal and vertical spacing between icons. The dialog box also allows you to modify iconic windows to vertically stagger alternate icons. By default, icons are not staggered. Changing the spacing and staggering settings of icons in your iconic navigator and tools windows reduces the possibility that the names of adjacent icons may obscure each other.

Figure 3-28. Iconic Navigator Setup Dialog Box



Procedure

1. In any Pyxis Project Manager window, choose **Setup > Preferences > Classic tab** from the pulldown menu.

The Iconic Navigator tabbed dialog box, shown in Figure 3-28, appears. Notice the vertical and horizontal icons, marked **A** and **B**. These show the actual current spacing between icons in the Pyxis Project Manager iconic windows.

2. To change the vertical spacing, in the vertical icon display, drag icon **B** away from or closer to icon **A**.
3. To change the horizontal spacing, in the horizontal icon display, drag icon **B** away from or closer to icon **A**.
4. To stagger your icons, click the “Stagger alternate icons?” prompt.
5. Execute the Iconic Navigator dialog box by clicking **OK**.

The Pyxis Project Manager immediately updates any iconic navigators and tools windows that are currently open, to reflect the new spacing.

Related Topics

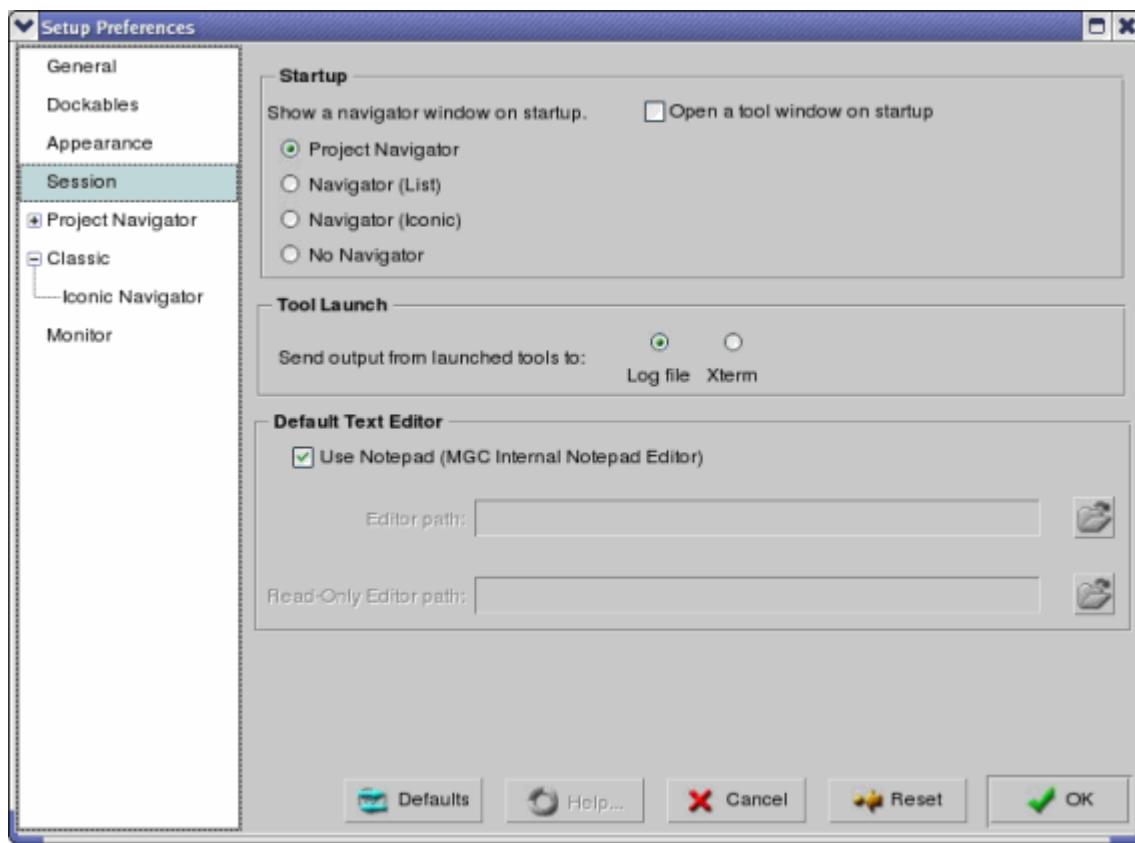
[\\$setup_iconic_window_layout\(\)](#)

Specifying Startup Windows

The Pyxis Project Manager provides you with the options to open a tools window and to open either an iconic or list navigator at invocation.

The creation of default tools and navigator windows at invocation is dependent upon the existence of startup files. If you do not have any startup files, the Pyxis Project Manager creates both a tools and navigator window at every invocation. If you have a custom startup file *dmgr.startup*, the settings in this file determine whether either or both of these windows are opened at invocation. If you do not have any startup files, except the default startup file *dmgr_default.startup*, the default startup file determines whether either or both of these windows are opened at invocation. In the absence of any startup files, both a tools window and an iconic navigator window are opened at invocation, by default.

Figure 3-29. Setup Preferences Dialog Box Session Tab



Procedure

1. In any Pyxis Project Manager window, execute the **Setup > Preferences > Session tab** pulldown menu item.
The Startup section is displayed in the top portion of the Session tab, as shown in [Figure 3-29](#).
2. To specify whether a tools window is to be opened at invocation, respond to the “Open a Tool Window on Startup?” prompt, by clicking this option.
3. To specify whether a navigator is opened at invocation, respond to the “Open a Navigator Window on Startup?” prompt, by clicking on the desired option.
 - The **Navigator (List Mode)** button specifies that a list navigator is opened at invocation.
 - The **Navigator (Iconic Mode)** button specifies that an iconic navigator is opened at invocation. (Default.)
 - The **No Navigator** button specifies that a navigator is not opened at invocation.

4. If you want the project navigator window also opened during the startup process, check the **Open the project navigator on startup** option. By default, this option is selected.
5. Execute the Startup Windows Setup dialog box by clicking **OK**.

Related Topics

[\\$setup_session_defaults\(\)](#)

[\\$setup_startup_windows\(\)](#)

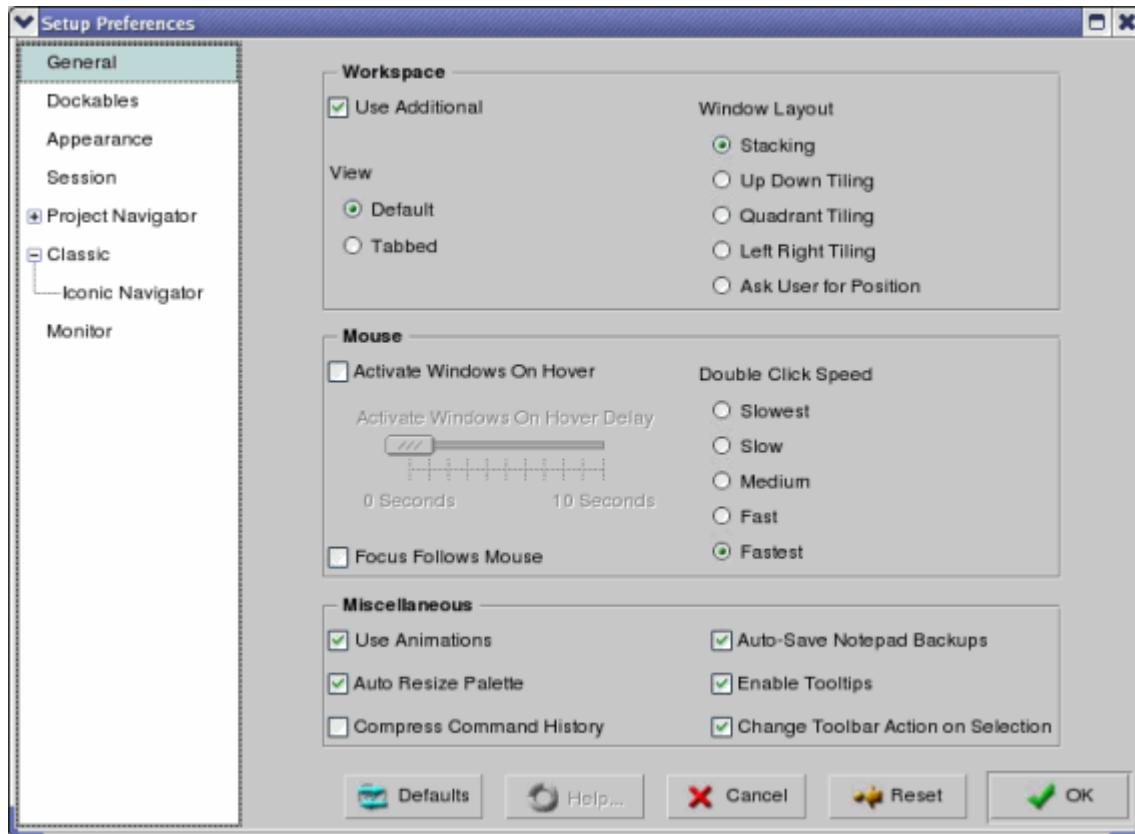
[Save Session Setup Settings](#)

Specifying Session Defaults

The Pyxis Project Manager has default settings that specify various characteristics of your session's graphical interface.

You can change your session setup to specify a different input device, a different double-click speed, a different window layout, or to specify that different window components are hidden or visible.

Figure 3-30. Setup Session Preferences Dialog Box General Tab



Procedure

1. In any Pyxis Project Manager window, execute **Setup > Preferences > General tab** from the pulldown menu. The Session Setup dialog box is displayed, as shown in [Figure 3-30](#).
2. To specify your double-click speed, respond to the "Double Click Speed" prompt by clicking on the desired option.

By default, the double-click speed is medium. You can specify that your double-click speed be any value in the range of Slow to Fast. The **Slow** button allows 2.25 seconds between the first and second click. The **Fast** button allows 0.25 seconds between the first and second click. The remaining three buttons are set at 0.5 second intervals.

3. To specify your window layout, respond to the “Window Layout” prompt by clicking on the desired option. By default, windows use Left Right tiling.
 - The **Stacking** button places windows on top of each other, with the most recently opened window on top.
 - The **Up Down Tiling** button places windows in either the top half or the bottom half of the session window area.
 - The **Quadrant Tiling** button places windows in one of the four quadrants of the session window area.
 - The **Left Right Tiling** button places windows in the left half or the right half of the session window area.
 - The **Ask User for Position** button allows you to specify a particular location for the window, by using the Select mouse button to frame the desired window placement location.
4. To specify whether menu bars, the message area, softkeys, and palettes are visible, click on each of the desired options. By default, the menu bars, the session title, and the message area are visible, and the softkey are and palettes are not visible.
5. The Refresh Heartbeat option automatically refreshes the project navigator. Specify the frequency at which you want the navigator to be refreshed by entering the number (of seconds) at the prompt.
6. Execute the Setup Preferences - General dialog box by clicking on the **OK** button.

Related Topics

[Save Session Setup Settings](#)

[\\$setup_session_defaults\(\)](#)

Specifying Navigator Filters

The navigator window displays the design objects that exist in its current directory.

By default, all design objects that exist in the active navigator directory are visible. However, the Pyxis Project Manager allows you to dynamically filter the design objects that you see in the navigator window. Using the **Setup > Preferences > Classic tab**, you can specify which design objects are visible by setting filters based on the design object's name and the design object's type. By default, navigator filters are set to include “*” which means that all files that do not begin with a “.” are displayed.

Specifying Filters for All Navigators	233
Setting Filters for the Active Navigator	235

Specifying Filters for All Navigators

The **Setup > Preferences > Navigator Filters > Classic tab** allows you to specify filter settings that apply to all of the navigator windows subsequently opened in the current session.

If you specify to save these filter settings the first time you execute a **Setup** menu item in the session, the filter settings are automatically saved in the Pyxis Project Manager default startup file *dmgr_default.startup*. This allows you to set your navigator filters once and reapply

When you execute the **Setup > Preferences > Classic tab**, a dialog box is displayed. The dialog box allows you to set inclusion or exclusion filters based on common strings in design object names and on common types among design objects.

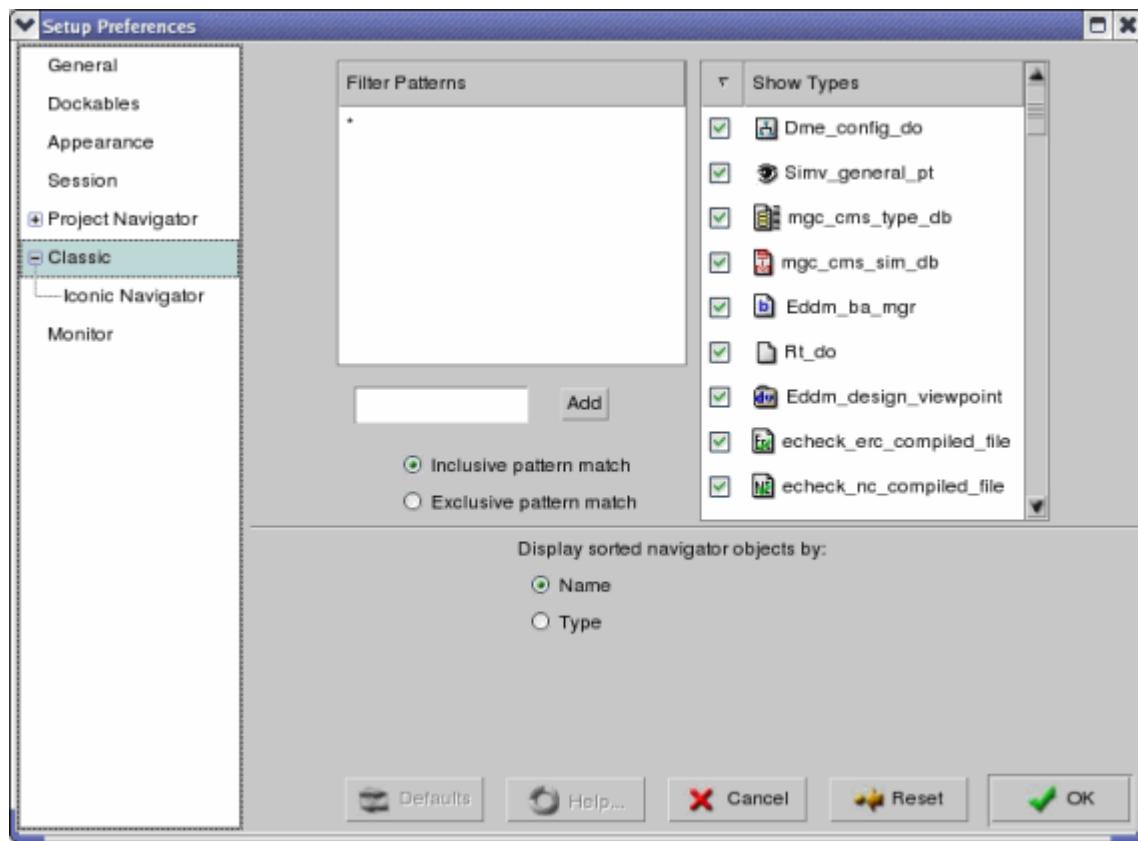
You can set any number of filters based on a design object's name by specifying string patterns using UNIX wild cards. You can set any number of filters based on a design object's type by selecting types in the type list area. Only objects which pass the name, type, and inclusion/exclusion criteria are displayed in the navigator window. You can also specify whether you want the design objects displayed in the navigator window alphabetically, or grouped first by type and then displayed alphabetically within each group.

After you execute this menu item for the first time in the session, the dialog box always displays with your most recent settings.

Procedure

1. In any Pyxis Project Manager window, execute the **Setup > Preferences > Classic tab > Filters tab**, which displays the dialog box shown in [Figure 3-31](#).

Figure 3-31. Setup Filters Tab



2. To set filters based on a design object's name, specify a string in the Pattern field.
The Pattern field allows you to specify any number of strings in the Pattern field. You can use UNIX wild cards to specify your strings.
3. To specify whether to include or exclude the design objects which pass the Pattern and type criteria, click on the **Inclusive Pattern Match** or **Exclusive Pattern Match** option.
 - The **Inclusive Pattern Match** button displays, in the navigator window, all of the design objects that match the strings specified in the Filter Patterns field that are also of one of the types specified in the "Show Types" field.
 - The **Exclusive Pattern Match** button displays, in the navigator window, all the design objects in the navigator directory *except* for those design objects that match the strings specified in the Filter Patterns field that are also of one of the types specified in the "Show Types" field.
4. To specify which types of design objects you want to include or exclude, click on the type names in the "Show Types" field, which by itself is a repository of all the types that you can specify to either include or exclude. When the field is empty, all types are either included or excluded.

5. You can add types to the Show Types field, by entering the type-name at the prompt next to the **Add** button, and then clicking the **Add** button.

i **Tip:** For more information about the Filter Options dialog box popup menu items, refer to the Notepad for *Pyxis User's and Reference Manual* on SupportNet.

6. To specify the order in which design objects are displayed in the navigator window, click on the **Name** or **Type** button in the “Display Sorted Navigator Objects By:” field.
 - The **Name** button sorts design objects alphabetically by name.
 - The **Type** button sorts design objects by type and, within each type group, alphabetically by name.
7. Execute the Setup Filter All dialog box by clicking OK.

Related Topics

[Setting Filters for the Active Navigator](#) [\\$setup_filter_all\(\)](#)

Setting Filters for the Active Navigator

The **Setup > Navigator Filters > Active Navigator** menu item allows you to specify filter settings for the active navigator.

Because the filters settings specified by this menu item apply only to the active navigator and are not saved in the Pyxis Project Manager default startup file, you can use this menu item to override the filter settings for a single navigator without changing the navigator filters specified in your default startup file.

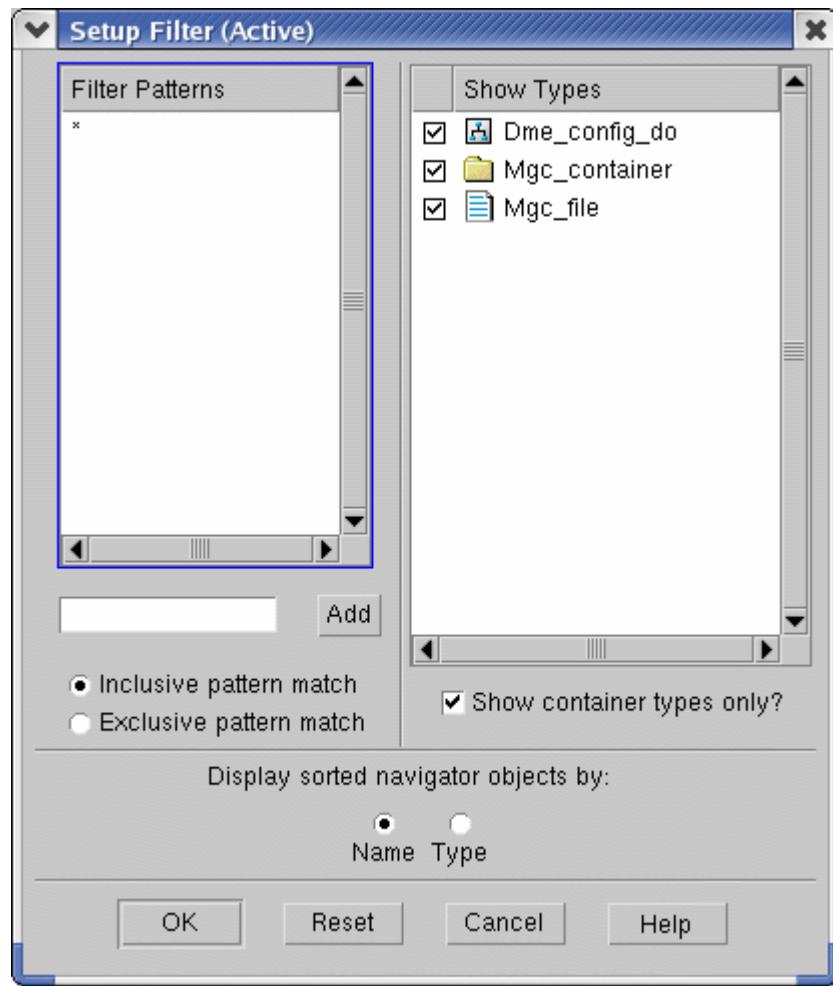
When you execute the **Setup > Navigator Filters > Active Navigator** from the active navigator window, the Setup Filter (Active) dialog box is displayed, as shown in Figure 3-32.

Because the dialog box is almost identical to the dialog box that you use to specify all navigator filters, you can use the same procedure described in the section “[Specifying Filters for All Navigators](#)” on page 233.

Procedure

1. You can specify your filter settings for the active navigator by performing steps 2 through 6 described in the previous procedure.
2. The only difference is that the Show Types field in the Setup Filter Active dialog box contains a **Show Container types only** button. This button allows you to choose whether the Show Types field displays all available types or only those types present in the current working directory of the active navigator. By default, the **Show Container types only** option is selected.

Figure 3-32. Filter Objects for the Active Navigator Dialog Box



3. When you execute the Setup Filter (Active) dialog box by clicking OK, the filters you specified in this procedure are set for the active navigator.

Related Topics

[Specifying Filters for All Navigators](#)

[\\$setup_filter_active\(\)](#)

Specify Configuration Monitoring

Both the configuration window and the configuration toolkit support real-time monitoring of configuration status for the Build Configuration, Copy Configuration, Delete Configuration, Release Configuration, and Change Configuration References operations.

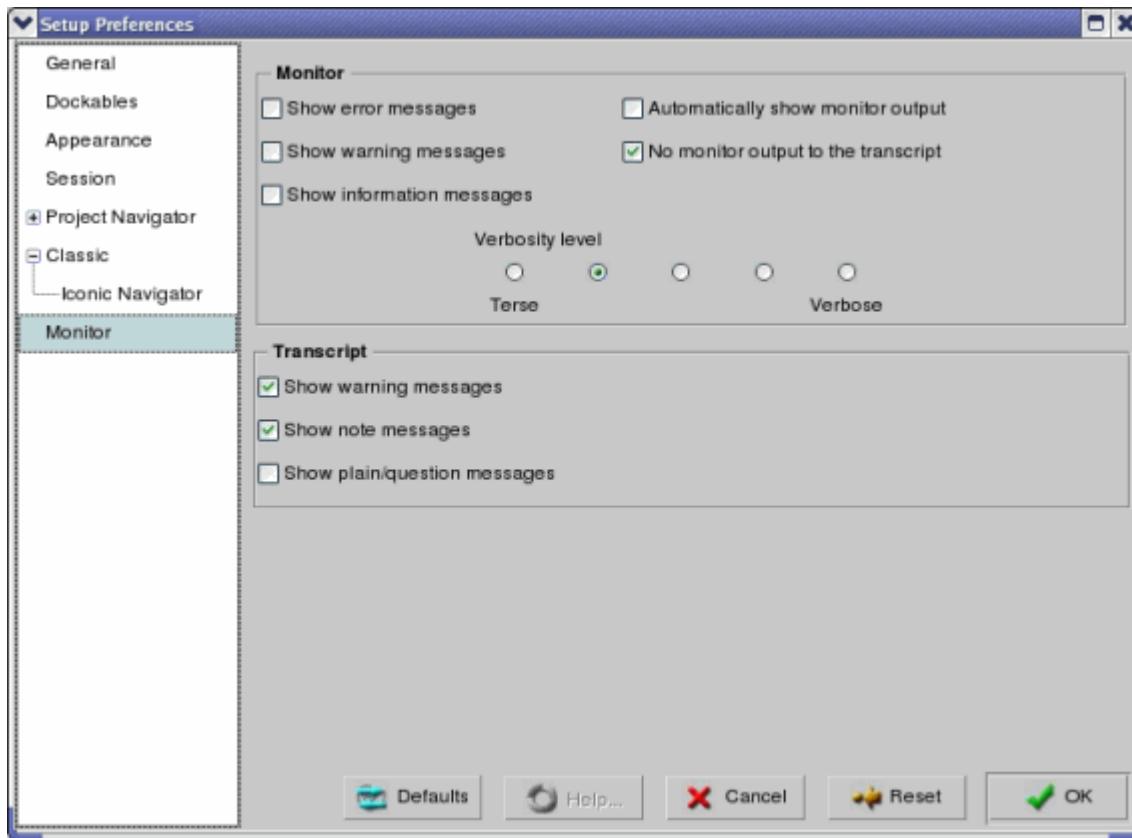
The Pyxis Project Manager provides you with options for customizing configuration monitoring behavior.

When you execute the **Setup > Preferences > Monitor tab** pulldown menu item, the Monitor Status Setup dialog box is displayed, as shown in [Figure 3-33](#). The dialog box fields provides options for the following:

- Filtering errors, warnings, and informational messages from the monitor report
- Specifying whether configuration monitoring information is output to the transcript window or to the monitor window
- Specifying whether the monitor window is visible or hidden
- Specifying the verbosity of configuration monitor reports.

By default, all errors, warnings, failures, and informational messages are not reported, the monitor status is not output to the monitor window, the monitor window is not hidden from view, and configuration monitoring verbosity is set to 2.

Figure 3-33. Monitor Status Setup Dialog Box



Related Topics

[Get Information About a Design Configuration](#)

[\\$setup_monitor\(\)](#)

Managing Designs

Use the configuration window and configuration object to manage your designs between application sessions.

i **Tip:** For more information on the configuration window and the configuration object, refer to the section “[Configuration Management Concepts](#)” on page 117.

Creating a Design Configuration	238
Viewing a Design Configuration	252
Opening an Existing Design Configuration	255
Get Information About a Design Configuration	256
Maintain the Configuration Hierarchy	259
Retargeting Configuration Entries	260
Copy a Design Configuration	262
Releasing a Design Configuration	266
Changing the References of a Design Configuration	272
Freezing and Unfreezing a Design Configuration	273
Lock a Design Configuration	274
Deleting a Design Configuration	275
Archive and Restore Designs	276
Copying Design Objects	280
Moving Design Objects	283
Deleting Design Objects	287
Changing References on Design Objects	288

Creating a Design Configuration

Before you can release a design, you must create a configuration that includes the design data and all related elements.

You may also have other reasons for creating a configuration, such as changing the references of a container and all of its contained design objects, or verifying that all references within a design point to objects within the design directory.

The following text describes the major steps that you perform to create a design configuration. These steps are:

- Opening a new, untitled configuration window.
- Manually adding design objects to the configuration. These additions are called *primary* entries.
- Specifying the *build rules* for including other related design objects. These related design objects are called *secondary* entries.

- Building the complete configuration. The build command traverses the containment hierarchy and, optionally, the reference network of each primary entry, thus adding secondary entries to the configuration.
- Saving the configuration to create a configuration object that you can use to recreate or to edit the configuration.

After building the configuration, you can perform operations in the configuration window that act on all configuration entries simultaneously. If you save the configuration after building it, you create a configuration object, which records which entries are part of the configuration and the build rules that you used to create the configuration. You can later recreate the configuration by opening the configuration object.

Opening a New Configuration	239
Adding a Primary Entry	239
Adding a Specific Design Object Version	240
Removing an Entry	241
Specifying the Build Rules	242
Using Maintain Hierarchy	245
Setting Target Paths	246
Build a Configuration	247
Including the Siblings of a Referenced Design Object	249
Saving a Configuration	251

Opening a New Configuration

The first step in creating a configuration object is opening a new, untitled configuration window.

Procedure

To open a new configuration window, choose a menu option:

- **Open Configuration > New** from the session window's popup menu.
- **Windows > Open Configuration > New** from the session window's pulldown menu.

Related Topics

[\\$open_configuration_window\(\)](#)

Adding a Primary Entry

After you open a new configuration or an existing configuration, you can add entries to and remove entries from the configuration. You can add only primary entries, but you can remove both primary and secondary entries.

You can add an object of any type to a configuration.

Primary entries are those that you add manually. Once added, the configuration window lets you automatically bring in related objects. As a result, primary entries are typically central to your design, and they often contain much of the rest of the design. For example, you might add, as a primary entry, a directory that contains schematics and simulation results.

Procedure

1. Choose **Add Entry** from the configuration window's popup menu.

An Add Configuration Entry object browser dialog appears, allowing you to navigate to a design object to add to the configuration.

2. Navigate to the desired design object and select it.
3. Click OK.

After using either of these methods, the new primary entry appears in the configuration window, according to the following rules:

- If the version already exists in the configuration as a primary entry, the configuration does not change.
- If the version already exists in the configuration as a secondary entry, the Pyxis Project Manager adds it as a primary entry, as well.

When you add a primary entry to the configuration, that entry's distinctive icon and full pathname are shown. In addition, the configuration window places a set of square brackets at the end of the entry's name, to indicate that only the current version of the design object is added as a configuration entry. To add other versions, you use the **Add Versions** menu item, which is described in the following text.



Tip: All configuration entries are single versions of a design object. As a result, when you operate on the configuration, you operate on only the single version of the design object that is represented in the configuration. For example, when you delete a configuration, you only delete the versions of the design object which are present in the configuration window, not the entire design object.

Related Topics

[\\$add_configuration_entry\(\)](#)

Adding a Specific Design Object Version

Each entry in a configuration represents only a single version of a design object.

When you add a primary entry, or when the Build Configuration command adds a secondary entry, the design object's current version is added. However, you can also add specific, design object versions, other than the current version, to the configuration. By selecting a primary entry and choosing **Add Versions**, you can choose one or more versions to add from a list in a dialog box.

Procedure

1. In the configuration window, select a primary or secondary entry.
2. Choose **Add Versions** from the configuration window's popup menu.
 - Adding version 4 adds a fixed entry to version 4 of the design object. As the design object evolves, the configuration still includes version 4.
 - Adding version [] adds the current version, and as the design object evolves, the configuration always includes the most current version.
3. Select the versions that you want to add.
4. Click **OK**.

The configuration window adds the version that you specified to its display. Whether you added a version of a secondary or primary entry, the added version is always added as a primary, and it always receives the default build rules.

You can select a previous version in the configuration and use **Add Versions** to add the current or any other version of the design object.

Related Topics

[Specifying the Build Rules](#)

Removing an Entry

You can remove a configuration entry any time before or after you build the configuration, and you can remove both primary and secondary entries.

When you remove a primary entry, that entry and all of its secondaries are removed. Removing an entry does not delete the corresponding design object. It simply removes the association that the configuration had with the design object.

Procedure

1. Select the entries that you want to remove.
2. Choose **Remove Entry** from the configuration window's popup menu.

Related Topics

[\\$remove_configuration_entry\(\)](#)

Specifying the Build Rules

After you add the primary entries to the configuration, you can specify the rules that the Build Configuration command follows when adding secondary entries.

These rules are referred to as *build rules*. By default, all primary entries have build rules that specify to include all design objects in the primary entry's containment hierarchy and reference network.

You can change a primary entry's build rules any time. In fact, as you build a configuration, you may repeatedly specify build rules, build the configuration, change the build rules slightly to include or exclude certain objects, and so on.

A design object's *containment hierarchy* is that object and everything that the directory tree beneath it contains. You cannot disable containment traversal with the build rules. However, the build rules do provide filters with which you can exclude certain objects.

A design object's reference network is all of the objects that can be reached from the object by traversing references. The build rules allow you to disable reference traversal, if desired. Build rules consist of the following elements, which you can combine, as needed, to create complex configurations:

- Expansion rule

The expansion rule determines whether or not references are traversed during the build. You can specify that the primary entry's entire reference network is traversed, or that no references are traversed.

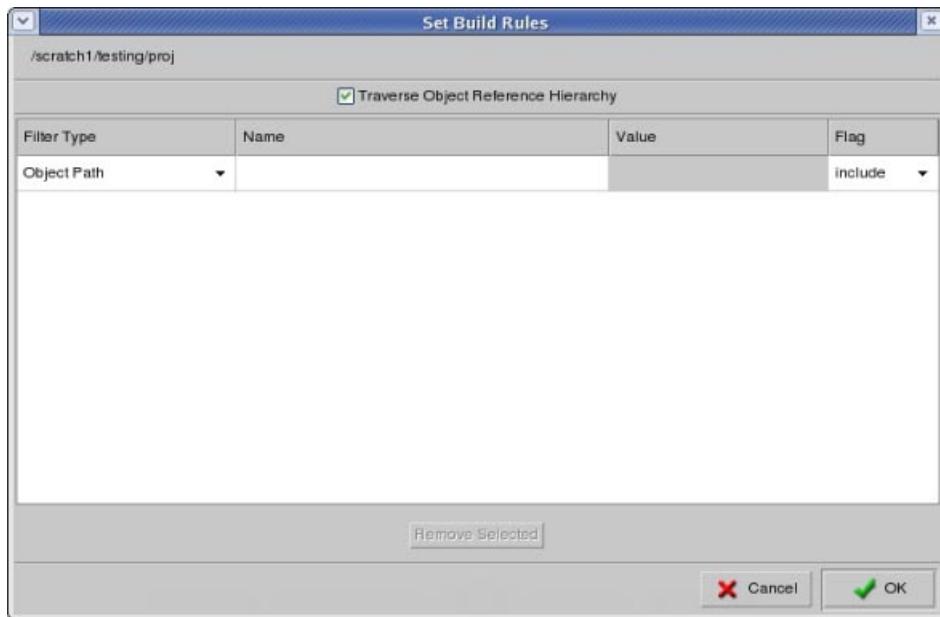
- Other filters

You can include or exclude secondary entries by name, by type, by properties that they hold, or by properties that the references that point to them hold.

Procedure

1. Select the primary entries whose build rules you want to set.
2. Choose **Set Build Rules** from the configuration window's popup menu.

If you have multiple primary entries selected, the Set Build Rules dialog box shown in Figure 3-34 appears, requesting **Reference Traversal** and **Filter** information. If you have a single primary entry selected, the dialog box appears with the current build rules for that entry displayed.

Figure 3-34. Set Build Rules Dialog Box

3. Specify the build rules you want to set for the selected primary entries, based on the following expansion rule and filter option descriptions:

- **Reference Traversal**

In addition to containment traversal, which is always enabled, you can specify that the Build Configuration command also traverses all references.

For example, if you turn reference traversal on, and if container *D* references objects *Y* and *Z*, objects *Y* and *Z* are added to the configuration as secondary entries. In addition, any objects that *Y* and *Z* reference or contain, and any objects that their referenced or contained children reference or contain, are added to the configuration as secondary entries. This traversal of all references and containment hierarchies continues until all associated containers and references are explored.

As with all configuration entries, only one version of a referenced object is included in the configuration. The version of the object to be included depends on the state of the reference that was traversed to find the entry:

- If the state of the traversed reference is current, the current version of the referenced object is added to the configuration.
- If the state of the traversed reference is fixed, the version to which the reference points is added to the configuration. If build traverses a fixed reference, then all other references traversed from that point in the reference network are treated as fixed, even if they are current references. That is, if build traverses a fixed reference from object *A* to version 10 of object *B*, and if object *B* has a current reference to object *C*, then the current version of object *C* is added to the configuration as fixed.

To understand what this means, assume that, at the time of the build, object *C* is at version 23. After building, you save the configuration and call it “fixed_config”. In the design in which *B* and *C* reside, *B* references the current version of *C*. As a result, as *C* evolves to version 27, *B* references version 27. However, if you open configuration “fixed_config”, you find that it contains version 23 of object *C*.

Remember, after build traverses a fixed reference, all references traversed from that point in the reference network are treated as fixed. In this way, build creates a stable “snap-shot” of any fixed portion of the design.

When traversing references, the build command detects and breaks infinite reference loops. An infinite reference loop occurs when Object *A* references Object *B*, and Object *B* references Object *A*. In this and all cases, a referenced design object appears only once in a configuration.

- **Other Filters**

You can use the filters presented below to include or exclude potential secondary entries. During a build, filters are applied to the objects that Build Configuration finds as it traverses containment and references.

To set your filters to include or exclude secondary entries in your configuration, use the following procedures in the Set Build Rules dialog box:

- i. Enter the pathname of the object. You can enter UNIX regular expressions as wild cards for the pathname as well. For example, `^.*lib$` matches every object name with the suffix “.lib”.
- ii. Click the **Include** or **Exclude** button depending on whether you want to include or exclude objects that match the specified pathname.
- iii. Enter the design object's type. You can use regular expressions as wild cards.
- iv. Enter the name and value of the object's property.
- v. You must enter the property name. Wild cards are not allowed, but entering the property value is optional.
- vi. Click the **Include** or **Exclude** button to either include or exclude those objects that hold a property with the specified name and value.
- vii. Enter the name and value of the reference property.
- viii. You must enter the property name, and wild cards are not allowed, but entering the property value is optional.
- ix. Click the **Include** or **Exclude** button to include/exclude those objects that the Build Configuration command reaches through a reference that holds a property with the specified name and value.

When you type into any of the filter text fields, notice that the dialog box automatically expands, allowing you to specify more filter patterns (inclusion or exclusion criteria) for that

particular filter category. The four filter categories, name, type, property, and reference property, work together to filter potential secondary entries out of the configuration, as follows:

- In any filter category, you can combine filters that include and exclude.
- If a design object *passes* (is not filtered out by) each of the four filter categories, Build Configuration adds it to the configuration.
- Empty filter categories, ones in which you do not specify any filters, always pass all design objects.

Related Topics

[\\$set_build_rules\(\)](#)

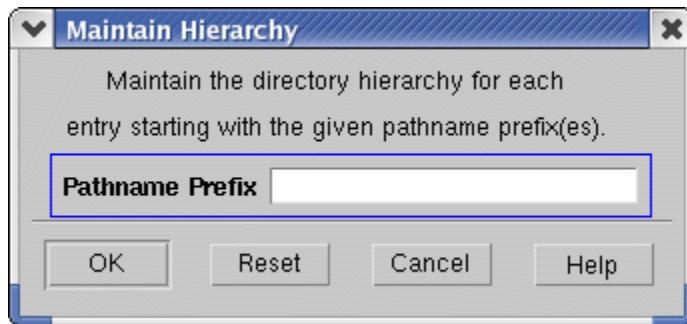
Using Maintain Hierarchy

You can maintain the directory hierarchy of your design objects by using the Maintain Hierarchy function from the Configuration Window popup menu.

Procedure

1. Choose **Maintain Hierarchy** from the configuration window's popup menu, or execute **Edit > Maintain Hierarchy** from the pulldown menu, which displays the Maintain Hierarchy dialog box, as shown in [Figure 3-35](#).

Figure 3-35. Maintain Hierarchy Dialog Box



2. Enter the prefix strings, which are the portion of the pathname you want to match.

The prefix string you determine is the portion of the pathname that is assessed on a directory by directory basis, if a match occurs that portion of the pathname is stripped off and the remaining directory structure is kept in tact upon doing a Copy or Release of the configuration. For example, if you have several entries that have the prefix, \$MGC_LIB and you specified that you wanted the hierarchy maintained from \$MGC_LIB downward, then you would enter \$MGC_LIB in the text field of the dialog box.

```
prefix string: $MGC_LIB
config entry: $MGC_LIB/and2
copy target: /usr/johnd/project
resulting config entry: /usr/johnd/project/and2
```

The prefix string in Maintain Hierarchy does not allow for wild card entries and processes the match exactly as it is specified. Because of this, be sure that your entries for prefix string are correct to avoid having unnecessary directories created.

3. Click **OK**.

By confirming the dialog box, each entry in the configuration is compared with the string you provided.

Related Topics

[\\$maintain_hierarchy\(\)](#)

Setting Target Paths

You can specify a target path other than the release destination path for any entry in the configuration using the configuration popup menu item **Set Target Path**.

Alternately, execute **Edit > Set Target Path** from the pulldown menu. When you perform either a release or copy of the configuration, the configuration entries for which you have specified target paths are released or copied to their target path, not to the target directory of the release or copy operation.

Procedure

1. In the configuration window, select the entry whose target path you want to specify.
2. Choose **Set Target Path** from the configuration window's popup menu.

A prompt bar displays in which you specify the target path for the selected entry.

3. Specify the target path in the Target Path field.

The target path is a pathname that you can specify for re-targetable configuration entries. A configuration entry is re-targetable if its parent container is not part of the configuration. You can check if an entry is re-targetable and view its target path, by executing the configuration popup or pulldown menu item **Report > Entry Info**.

If you specify a target pathname using an absolute pathname, the target pathname takes precedence over the release or copy destination. If you specify a target pathname using a relative pathname, the target path is concatenated to the end of the destination pathname of the release or copy. If you specify a target pathname as the empty string "", the

entry's target path is the default which is the leaf name at the target destination of the release or copy.

The target pathname is interpreted differently, depending on whether you have a single or multiple configuration entries selected. If you execute this menu item with a single configuration entry selected, the specified target path is assumed to be the entire target path. If you execute this menu item with multiple configuration entries selected, the specified target path is assumed to be a directory to which the leaf name of each entry is appended.

4. Click **OK**.

The specified target path is set for the selected configuration entries. You can view the target path for a configuration entry by selecting the entry and executing the configuration popup or pulldown menu item **Report > Entry Info**.

Related Topics

[\\$set_target_path\(\)](#)

Build a Configuration

After you set your build rules for each primary entry, and specify target paths for any entry that you want to retarget, you can build the configuration.

To build a configuration, choose **Build** from the configuration window's popup menu.

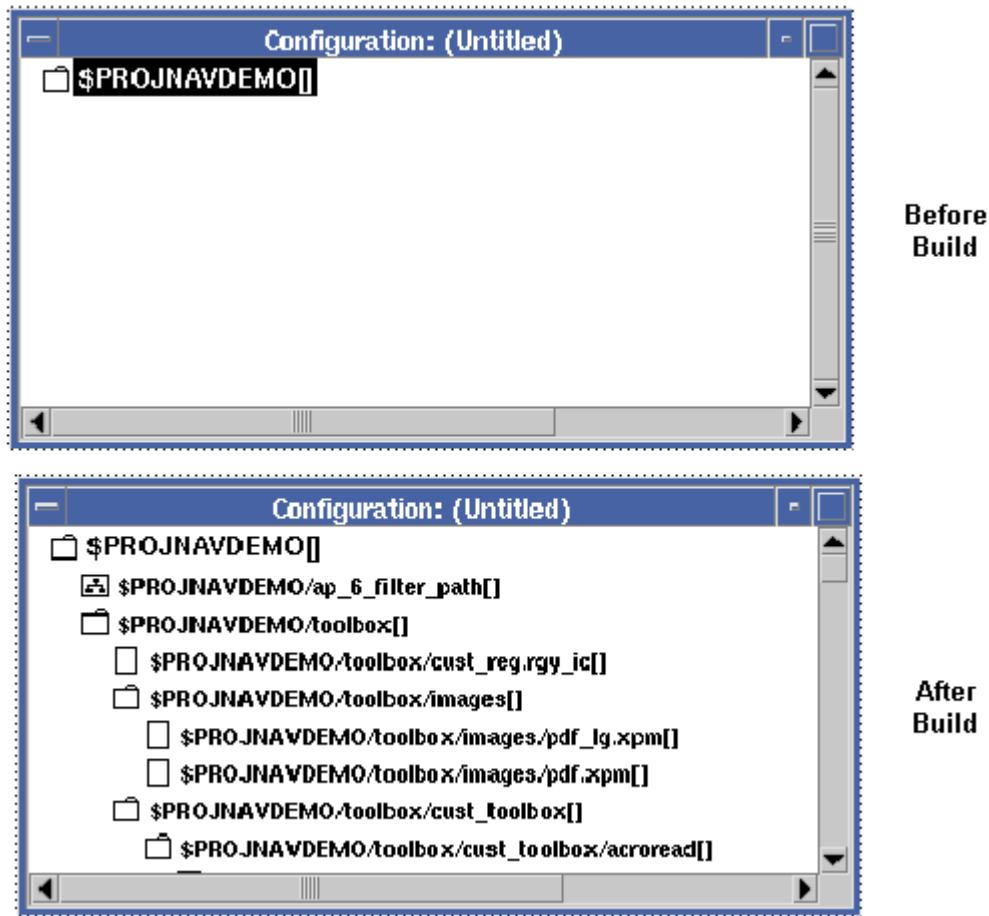
The Pyxis Project Manager uses the build rules specified for the primary entries and creates a configuration. The build command reports real-time status in the monitor window, during the course of the operation.

If your session setup values specify to show the monitor window, the monitor window is opened when you execute this function. When the operation is complete, the monitor window remains visible until you remove it from view.

You can remove the monitor window from view, by executing the monitor window's popup menu item **Hide Monitor**. This command removes the monitor window and re-displays the configuration window containing the new configuration, which appears similar to Figure 3-36. You can return to view the report in the monitor window, at any time prior to the execution of another configuration operation, by executing the configuration window's popup menu item **Show Monitor**.

If the build encounters an error, the operation does not terminate. The operation continues to build as much of the configuration as it can and displays error messages for those entries that fail. You can interrupt the build operation at any time by pressing the Ctrl and Backslash keys together.

Figure 3-36. Configuration Window, Before and After Building



The build operation automatically resolves any target conflicts that exist in the configuration. For each conflicting *re-targetable entry*, that is a configuration entry whose parent is not part of the configuration, the operation includes the entry's parent container as a secondary entry; if the addition of the parent entry creates a new conflict, the parent's parent container is added. This resolution process of adding parent containers continues until all conflicts are resolved.

Conflict resolution occurs after the normal build process is complete, so that no additional entries, other than the parent entries, are added. Conflict checking is based on the value of the target path of the re-targetable entry. This means that if you introduce conflicts by manually changing the target path of any of the primary entries, these conflicts are resolved as well.

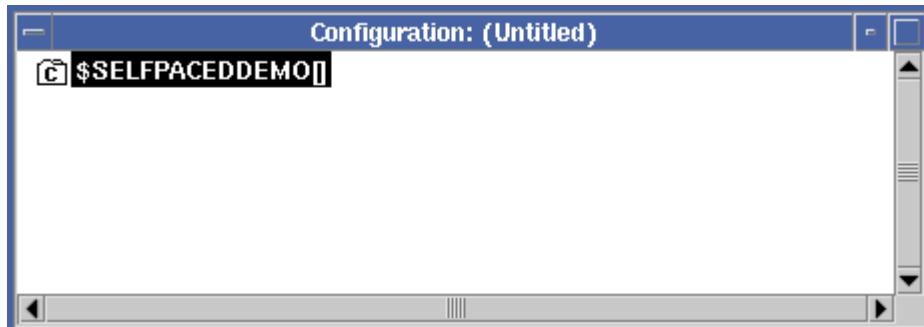
The following example shows how automatic conflict resolution works.

Assume that a configuration contains the primary entry `$PROJ_DESIGN/d2/design1`, but does not contain its parent `$PROJ_DESIGN/d2`. When you release the configuration to the destination directory `$PROJ_DESIGN/d1/release`, the entry is placed in the target directory as a leaf name only, creating the entry `$PROJ_DESIGN/d1/release/design1`.

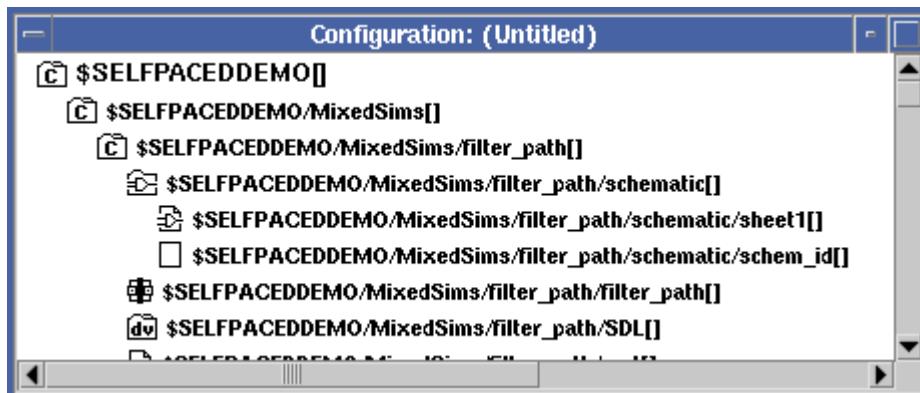
Now, assume that the configuration also contains the primary entry `$PROJ_DESIGN/d1/design1` and does not contain `$PROJ_DESIGN/d1`. Without the resolution capability of the build operation, a release operation would encounter the second primary entry and attempt to create the entry `$PROJ_DESIGN/d1/release/design1` again, creating a conflict in the release directory even though both of the design objects are distinct design objects. To prevent this from occurring, the build operation recursively adds the parent container of each conflicting re-targetable entry to the configuration, until all conflicts are resolved. Figure 3-37 illustrates the addition of the entry's parent containers to resolve potential target conflicts during a release or copy operation.

Figure 3-37. Leaf Resolution, Before and After Building

Before



After



Related Topics

[Change Your Session Setup](#)

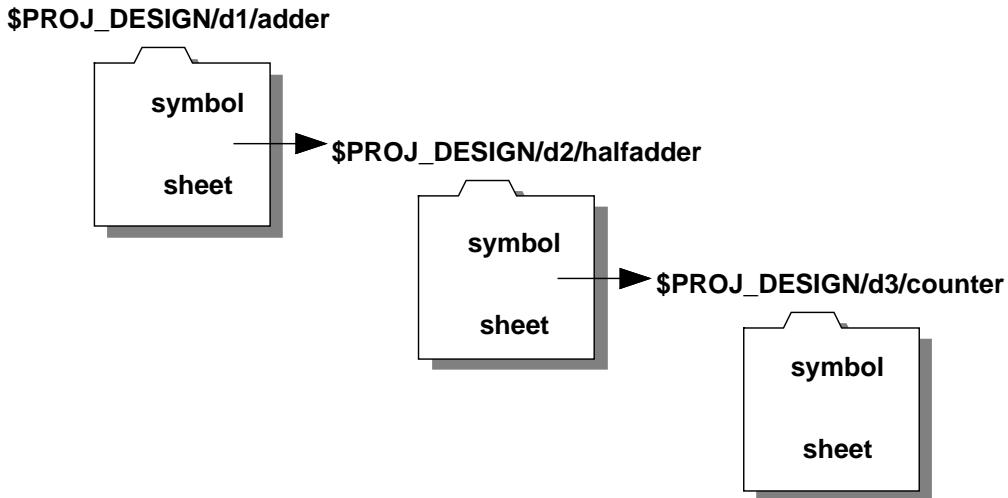
[Creating a Configuration](#)

Including the Siblings of a Referenced Design Object

The configuration window allows you to create complex configurations.

However, it has a potential limitation because it does not automatically include the parent or siblings of a design object that is added to the configuration as a result of a reference traversal, unless there is a target conflict, as discussed in the previous text. *Siblings* are those design objects that reside in the same directory as the referenced object. For an example of this limitation, refer to Figure 3-38.

Figure 3-38. Referencing Objects Inside of the Container



When releasing this design, you probably want all of the objects shown in the picture to be included in the release. However, under normal circumstances, with reference traversal enabled, the Build Configuration command only includes the following in the configuration:

- `$PROJ_DESIGN/d1/adder`
- `$PROJ_DESIGN/d1/adder/symbol`, and its contents and references, if any
- `$PROJ_DESIGN/d1/adder/sheet`, and its contents and references
- `$PROJ_DESIGN/d2/halfadder/symbol`, and its contents and references, if any.

Missing are the `$PROJ_DESIGN/d2/halfadder` directory, the sheet contained within, and the entire `$PROJ_DESIGN/d3/counter` directory, including its sheet and symbol.

To solve this limitation, some design object's types hold a certain property that the configuration window uses to include the missing items. This property is named *Dme_config_include_container*. If a design object that you include by reference is an instance of a type that holds this property, the parent of that design object is automatically included in the configuration, regardless of build rules. Because containment traversal is always enabled, that parent's children (the referenced object's siblings) are also potentially included in the configuration. The build rule filters are applied to these children.

If the type of the *symbol* design objects in Figure 3-38 has this property, all of the objects shown are potentially included in the configuration. The entire `$PROJ_DESIGNX/d2/halfadder` directory, including its contents, is included, and the contents and references of its children are explored for more objects. Because `$PROJ_DESIGNX/d3/counter/symbol` is an instance of a type that holds the property, the entire `$PROJ_DESIGNX/d3/counter` directory and its contents are potentially included.

To determine if a design object's type holds the *Dme_config_include_container* property, perform the following steps:

1. Determine the type name of the design object whose type you suspect has this property, by performing the following steps:

- a. Navigate to the design object.
- b. Select the design object.
- c. Execute **Report > Object Info** from the navigator's popup or pulldown menu.

A read-only window appears, displaying, among other items, the design object's type.

- a. Note the type name.

2. Activate the session window, and then press the Command key to display the floating command line.
3. Type in the **Open Types Window** command and press the Return key.

The “Type Manager: Known Type Reps” window appears. This window displays a sorted list of all of the design object types known to the Pyxis Project Manager, with no duplicates.

4. Select the type whose name you noted in step 1.d.
5. Choose **Report Type Info** from the types window's popup menu or **Report > Type Info** from the types window's pulldown menu.

A Type Information Report window appears and displays, among other information, the properties that the type holds. If one of the properties is named *Dme_config_include_container*, the automatic container inclusion features works for instances of this type.

Related Topics

[\\$report_type_info\(\)](#)

[\\$open_types_window\(\)](#)

Saving a Configuration

After you build the configuration and are satisfied that it contains the entries that you want, you can save it for later use.

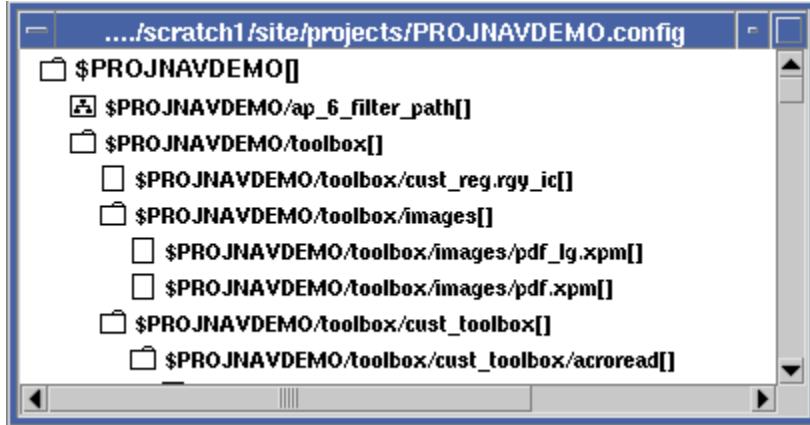
For a view of a saved configuration, refer to Figure 3-39. If you do not save it, it is simply discarded, when you exit the configuration window.

Procedure

1. Choose **Save As** from the configuration window's popup menu, or execute **File > Save As** from the pulldown menu, which displays a prompt bar.
2. In the Path field, enter the pathname to which you want to save the configuration and press the Return key.

The name to which you saved the configuration appears in the title bar of the configuration window. Also, as shown in Figure 3-39, the resulting configuration object is automatically added to the configuration.

Figure 3-39. Saved Configuration, with Configuration Object Added



The fact that the configuration object is automatically added is especially useful when releasing a configuration. Because the configuration object is now part of the configuration, it is included in the release. As a result, to view the contents of the release, you just open the configuration object in the target directory. In addition to recording the contents of a configuration, a released configuration object also records important information about the release, such as the time and owner of the release, and the source location of the released objects.

Related Topics

[Getting Information about a Release](#)

[\\$save_configuration\(\)](#)

Viewing a Design Configuration

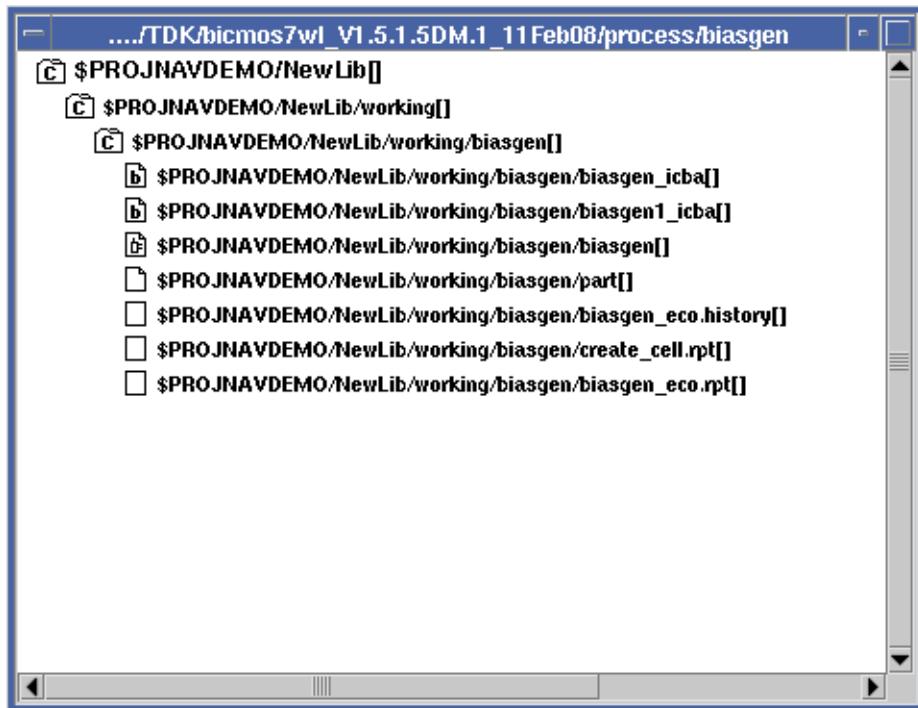
The Pyxis Project Manager provides you with the ability to view a design configuration by containment or hierarchically.

View the Containment Hierarchy	252
View Primary and Secondary Entries	253

View the Containment Hierarchy

Viewing the configuration's *containment hierarchy* is the default mode for the configuration window.

This view displays the containment hierarchy (which objects are contained by which objects) by using successive levels of indentation. Figure 3-40 shows an example of a window containing entries of a containment hierarchy.

Figure 3-40. Viewing the Containment Hierarchy

When you view the containment hierarchy of a configuration, the primary and secondary relationship of the configuration entries is not shown. An object whose name appears flush left in the list window may be a secondary object, and an indented name may be a primary object.

Viewing a configuration by containment hierarchy is very helpful when you need to identify re-targetable objects. To be re-targetable, an entry's parent directory must not be in the configuration. Thus, only those objects that are left-justified are re-targetable.

To restore the primary/secondary view, choose **View Primaries** from the configuration window's popup menu.

Related Topics

[Retargeting Configuration Entries](#)

[\\$viewContainmentHierarchy\(\)](#)

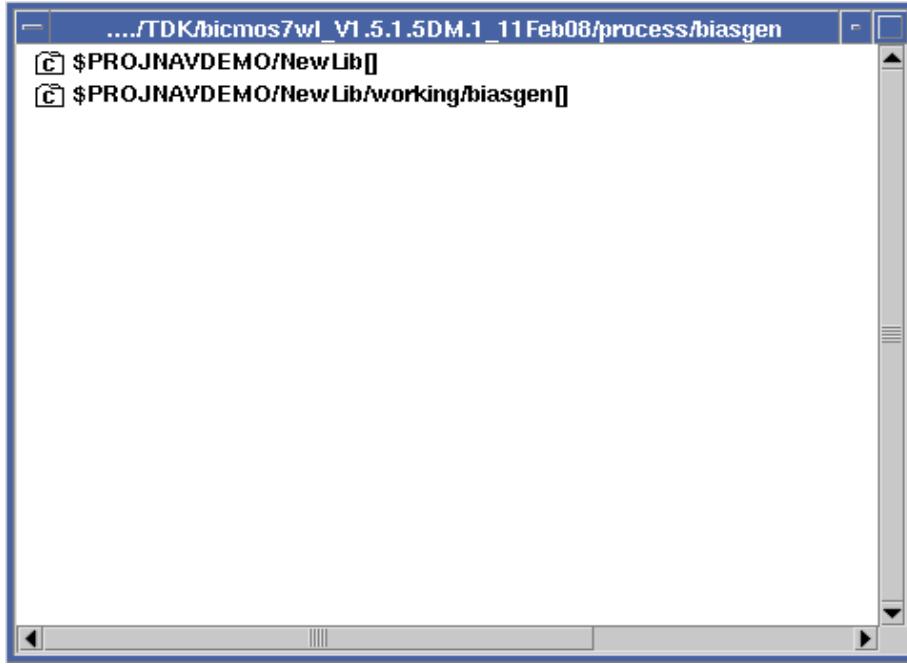
View Primary and Secondary Entries

You can view the primary and secondary relationships in your configuration by choosing **View Primaries** from the configuration window's popup menu.

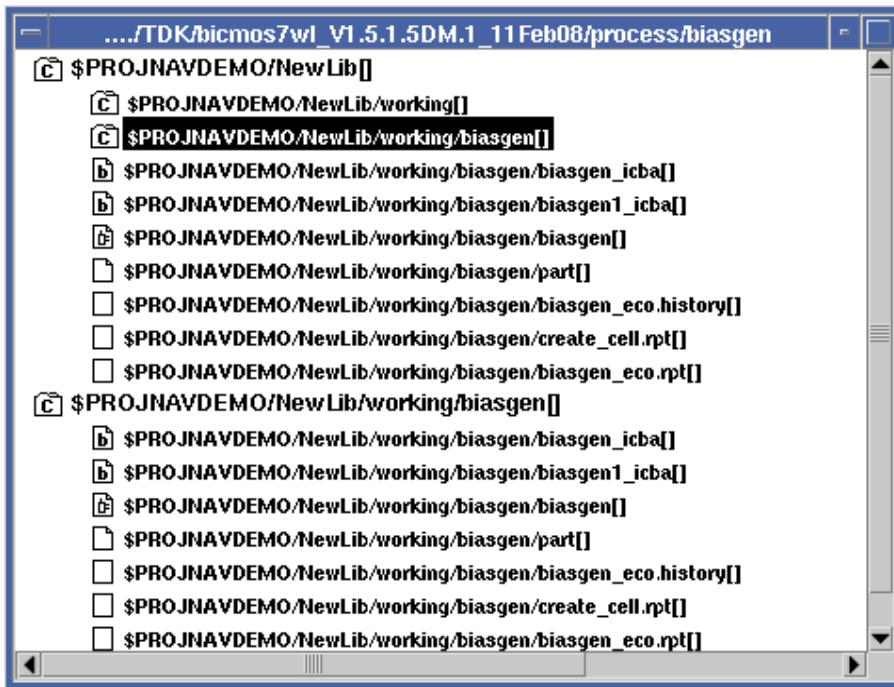
Because several primary entries can have similar build rules, a single design object can appear as the secondary entry of several primary entries. When you select one of these duplicate secondaries, they all highlight, indicating that these multiple secondaries only represent a single design object.

You can hide a configuration's secondary entries, by choosing **Hide Secondaries** from the configuration window's popup menu, as shown in [Figure 3-41](#). The secondary entries then are hidden and only the primary entries are visible in the configuration window.

Figure 3-41. Viewing the Primary Hierarchy



You can restore a configuration's secondary entries, by choosing **View Secondaries** from the configuration window's popup menu. Both the primary and secondary entries are then visible in the configuration window. As shown in [Figure 3-42](#), the primary entries are left-justified, with each primary's secondary entries indented, to the right, beneath it.

Figure 3-42. Viewing the Secondary Hierarchy

Related Topics

[\\$view_primary_hierarchy\(\)](#)

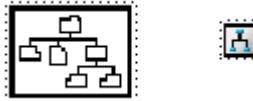
Opening an Existing Design Configuration

After you create a configuration and save it, you can open the resulting configuration object, allowing you to change the configuration or to perform operations on it.

Procedure

1. In a navigator, find the configuration object that you want to open.

Figure 3-43 shows the configuration object's large and small icons.

Figure 3-43. Large and Small Configuration Object Icons

2. Select the configuration object that you want to open.
3. Execute **Open > Configuration Window** from the navigator's popup menu.

To open a previous version of a configuration object, perform the following steps:

1. In a navigator, find the configuration object whose previous version you want to open.
2. Select the configuration object.
3. Choose **Report > Show Versions** from the navigator's popup or pulldown menu.

A version window appears, displaying the versions of the selected configuration object.

4. Double-click the version of the configuration object that you want to open.

Every configuration object version that is displayed in the version window is considered a *previous version*. Even if you double-click the highest numbered version, which is the current version number, you open a configuration window in read-only mode.

You cannot edit a configuration in any of the following circumstances:

- If the configuration object's file system protection does not allow you to edit it.
- If you opened a previous version of a configuration.
- If you opened a configuration object that has been released. As with all released design objects, the software prevents you from creating new versions of a released configuration object.

In each case, the configuration window senses that you opened a non-editable object, and it greys out all menu items that change the configuration in any way. In addition, it greys out the **Global Operations > Release** menu item. To enable editing on a read-only or released configuration object, you must copy the object to a new location and edit the copy.

Related Topics

[\\$release_object\(\)](#)

Get Information About a Design Configuration

The configuration window provides access to several different kinds of configuration monitoring information. Depending upon which menu option you select, you can report information about a configuration's status, about the status of a configuration's entries, about the status of a configuration entry's references, or about the integrity of a configuration's entries.

All configuration monitoring information is reported in real-time and, by default, is displayed in the active configuration's monitor window. If your session is not setup to automatically display the monitor window during configuration operations, you can view the configuration monitor window by executing the configuration window's popup menu **Show Monitor**. When you finish viewing the monitor window, you can return to the active configuration window by executing the monitor window's popup menu item **Hide Monitor**. By default, the monitor window is cleared upon the execution of each new configuration operation. If you have not executed a new configuration operation since the last monitoring operation, you can review the monitoring information in the monitor window again, by executing the **Show Monitor** menu item.

The Pyxis Project Manager provides you with options for customizing configuration monitoring characteristics through the pulldown menu item **Setup**, which is available in all windows. By default, the Pyxis Project Manager reports configuration monitoring status without any filters, sends monitor output to the monitor window, hides the monitor window during configuration operations, and outputs relatively terse monitor reports. The following procedures are based on these default settings. You can change any one of these default setting by executing the `$set_monitor_flag()` function.

i **Tip:** For information on customizing your configuration monitoring characteristics, refer to the section “[Specify Configuration Monitoring](#)” on page 236.

Reporting a Configuration's Status	257
Reporting Configuration Entry Status	258
Reporting Configuration Entry References	258
Reporting Configuration Entry Verification	259

Reporting a Configuration's Status

Report information about the active configuration.

Procedure

1. Open the configuration object about which you want to report information.
2. Execute **Report > Configuration Info** from the configuration window's popup or pulldown menu.

The configuration monitor window appears, displaying the following information:

- Configuration Object—The configuration object's absolute pathname.
- Lock Status—The configuration is locked or not locked. Locking a configuration prevents others from accessing the design object versions that are part of the configuration.
- Configuration Build Status—The configuration's build status includes the state of the configuration in respect to the last successful build operation. The three possible configuration build status messages and their meanings are listed below:
 - “The last Build was completed successfully without errors.”

A successful build has been performed since the configuration was locked. No changes have been made to either the build rules or the configuration entries since the last successful build.

- “A successful Build has not been done since being locked.”

A successful build has not been accomplished since the Configuration was locked.

- “The configuration Build is either out-of-date or incomplete.”

The configuration has never been build. A successful build has been performed since the configuration was locked. Changes have been made to either the build rules or the configuration entries since the last successful build.

- Freeze Status—The configuration is frozen or not frozen. Freezing a configuration prevents the version depth mechanism from deleting those entries (design object versions) that are beyond the version depth of each design object that is part of the configuration.
- Change Status—The configuration has changed or has not changed since the last time you saved it. A change is defined as changing a build rule, adding or deleting an entry, or building the configuration.

Related Topics

[\\$report_configuration_info\(\)](#)

Reporting Configuration Entry Status

Report information about the entries in the active design configuration.

Procedure

1. Open the configuration object for that configuration.
2. Select the entries whose status you want reported.
3. Execute **Report > Entry Info** from the configuration window’s popup or pulldown menu.

The configuration monitor window appears, displaying information about the entry.

Related Topics

[Retargeting Configuration Entries](#)

[\\$report_entry_verification\(\)](#)

Reporting Configuration Entry References

Report information about the a configuration entry’s references.

Procedure

1. Open the configuration object for that configuration.

2. Execute **Report > References** from the configuration window's popup or pulldown menu, which displays a prompt bar.

3. Enter the text pattern and select the appropriate reference switch.

The pattern string is matched against the reference pathnames. References whose pathnames contain a matching string are filtered from the report. The reference switch indicates whether all references are reported or only broken references are reported.

4. Execute the prompt bar.

The configuration monitor window appears, displaying information about the entry, including an absolute pathname and version number.

Related Topics

[\\$report_configuration_references\(\)](#)

Reporting Configuration Entry Verification

Report configuration entry verification information.

Procedure

1. Open the configuration object for that configuration.
2. Execute **Report > Entry Verification** from the configuration popup or pulldown menu.

The configuration monitor window appears, displaying the total number of entries, as well as the number of broken and out-of -date entries.

You can use Pyxis Project Manager functions to develop your own customized configuration reports. You can use the `$show_monitor()`, `$hide_monitor()`, `$$writeln_monitor`, and `$$clear_monitor()` functions to manipulate the configuration monitor window.

Related Topics

[\\$show_monitor\(\)](#)

[\\$\\$clear_monitor\(\)](#)

[\\$hide_monitor\(\)](#)

[\\$report_entry_verification\(\)](#)

Maintain the Configuration Hierarchy

In addition to being able to reset the target path of a specific entry in the configuration, there are times when you may want to preserve the hierarchy of the entries, when you do a copy or release function.

This is a typical scenario for ASIC vendor groups who want to improve the support for maintaining the source area structure in the destination directory. By using the Set Target

Hierarchy command you can set multiple prefix and target pathnames for entries in a given configuration.

Related Topics

[\\$set_target_path\(\)](#)

Retargeting Configuration Entries

When you copy or release a configuration, you may want to specify a different target path, for some configuration entries, than the release or copy destination. The configuration window allows you to specify that particular configuration entries be copied to a location different from the target path of the operation.

Procedure

1. In the configuration window, select the re-targetable entries whose target path you want to change.

Re-targetable entries are those entries whose parent directories are not also entries in the configuration.

2. Choose **Set Target Path** from the configuration window's popup menu, which displays a prompt bar.

You can use **Set Target Path** to specify a new leaf name for your configuration entry. To do this, you simply specify a file system location that does not exist. During a copy or release, your configuration entry is copied to that name. In fact, if you mis-type while specifying the alternate target, you can easily specify a new leaf name without intending to do so.

3. Enter a pathname in the Target Path field, and execute it.

The Pyxis Project Manager uses the following target pathname rules:

- If you supply an absolute pathname, the Pyxis Project Manager uses that pathname as the target destination for that object.
- If you supply a relative pathname, as shown in the Set Target Path prompt bar, the relative pathname is concatenated to the end of the copy or release destination pathname.

When you copy or release the configuration, the selected entries are not copied to the destination that you specify for the operation. Instead, they are copied to the absolute pathname that you entered in the prompt bar, or to the absolute pathname that results from the concatenation of the operation's destination pathname with the relative pathname.

The target pathname is interpreted differently, depending upon whether you have a single re-targetable entry or multiple re-targetable entries selected. If you execute the Set Target Path command with a single re-targetable entry selected, the specified target path is assumed to be the entire target path. If you execute the Set Target Path command with multiple re-targetable entries selected, the specified target path is assumed to be a directory to which the leaf name of each entry is appended.

The following examples illustrate how re-targeting enables you to select specific entries in your configuration to be targeted for a destination other than the destination you specify during copy and release operations.

- Example 1

You enter *d1/xxx* in the prompt bar, and click **OK**. This pathname is a relative pathname because it does not begin with either a “/” or a “\$”. During configuration release, you specify the soft path *\$PROJ_DESIGN/release_dir* for the destination. After the release occurs, the pathname of *xxx* is *\$PROJ_DESIGN/release_dir/d1/xxx*.

- Example 2

You enter *d2/xxx* in the prompt bar, and click **OK**. This pathname is a relative pathname because it does not begin with either a “/” or a “\$”. During configuration release, you specify the hard path */user/d2/project/release_dir* for the destination. Because */user/d2/project* is a physical path to the soft prefix *\$PROJ_DESIGN* in your location map, when the release occurs, the hard path */user/d2/project/release_dir* is resolved to its soft path equivalent *\$PROJ_DESIGN/release_dir* and the pathname of *xxx* is *\$PROJ_DESIGN/release_dir/d2/xxx*.

- Example 3

You enter *\$PROJ_DESIGN/d3/xxx* in the prompt bar, and click **OK**. During configuration release you specify */tmp* for the destination. The final pathname of *xxx*, after the release occurs, is *\$PROJ_DESIGN/d3/xxx*.

The *\$PROJ_DESIGN/d3/xxx* path you entered in the prompt bar is an absolute pathname, so the destination directory you specified during configuration release, */tmp*, is ignored for design object *xxx*, and the target pathname you specified is the destination directory for *xxx*. The target field always takes precedence when it contains an absolute pathname.

- Example 4

When you retarget a directory, and release or copy the configuration, the directory's contents also go to the alternate destination. The Pyxis Project Manager places the contained objects relative to the alternate target. In Example 1, if *d1/xxx* is a directory that contains *yyy*, after a release or copy, *yyy*'s new pathname is *\$PROJ_DESIGN/release_dir/d1/xxx/yyy*. In Example 2, if *d2/xxx* is a directory that contains *yyy*, after a release or copy, *yyy*'s new pathname is

`$PROJ_DESIGN/release_dir/d2/xxx/yyy`. In Example 3, after a release or copy, `yyy`'s new pathname is `$PROJ_DESIGN/d3/xxx/yyy`.

You can restore the default pathname at any time, by performing the following steps:

1. In the configuration window, select the entries you want to restore to the default target pathname.
2. Execute **Set Target Path** from the popup menu and leave the Target Path field blank.
3. Click **OK**.

As a result of doing this restoration, the selected entries are copied or released to the same location as the specified destination directory for those operations.

Although you can attempt to retarget any entry, for the retargeting to work properly, the entry must be *re-targetable*. An object is re-targetable if its parent directory is not part of the configuration. For example, although `$PROJ_DESIGN/d1/sch` and `$PROJ_DESIGN/d1/sch/test1` may both be primary entries, you cannot specify a different destination directory for `$PROJ_DESIGN/d1/sch/test1` because its parent is part of the configuration. If you retarget this entry, the alternate path is simply ignored.

The Pyxis Project Manager provides an easy way to view re-targetable and non-re-targetable entries by using the View Containment Hierarchy command.

Related Topics

[\\$set_target_path\(\)](#)

[\\$viewContainmentHierarchy\(\)](#)

Copy a Design Configuration

When you copy a configuration, you create copies of each design object that is part of the configuration. The location to which the design objects are copied depends on both the destination directory you specify and the alternate target pathname that you may have assigned to individual entries.

i **Tip:** For more information about specifying alternate pathnames, refer to the section [“Retargeting Configuration Entries”](#) on page 260.

The Copy Configuration command works like the navigator's Copy Object command, except that it operates on each configuration entry. This can be accessed through the configuration window's popup menu under **Global Operations > Copy**, which brings up the Copy Configuration dialog box.

The Copy Configuration command provides the following features:

- When you copy a configuration, the leaf names of the configuration entries are copied to the destination. In other words, if the configuration contains entry `$PROJ_DESIGN/d2/schematic`, and the destination of the copy is `$PROJ_DESIGN/d1`, the Copy Configuration command creates design object `$PROJ_DESIGN/d1/schematic`.

The exception to this rule is the configuration entry whose parent directory is also in the configuration. For example, if the *schematic* design object from the previous example is a container that contains design object *sheet*, the Copy Configuration command creates design object `$PROJ_DESIGN/d1/schematic/sheet`.

- Occasionally, you may specify a destination directory that contains design objects whose name conflicts with the names of the configuration entries that you want to copy.

To handle this case smoothly, the Copy Configuration command allows you to specify, prior to the copy, how the Pyxis Project Manager handles conflicts in the destination directory. You can either cancel the entire copy operation or replace the conflicting design object with the copy.

The Copy Configuration command allows you to specify a preview option, to help prevent replacing important data. You can preview the entire copy operation prior to performing it.

- When you copy a directory in the navigator, all of the directory's contents, to the bottom of the tree, are also copied. However, the Copy Configuration command only copies the displayed contents of the configuration window. In other words, if your configuration includes a container or directory, when you copy the configuration, that container or directory is copied, but *only the contents shown in the configuration window are also copied*.
- When you copy a configuration, design object versions are renumbered in the destination directory, starting at version one, if the design object does not currently exist in the destination directory. If the design object does exist in the destination directory, the new version is layered on top of the existing versions and renumbered to the next sequential number. For example, if you copy version 5 of a design object, to a destination directory containing versions 1 and 2 of the same design object, the new version in the destination directory is renumbered to version 3.
- The copy configuration command reports and displays status information, such as errors and warnings, in real-time to the monitor window.

If your session setup values specify to show the monitor window, the monitor window is opened when you execute this function. To hide the monitor window, execute the configuration's popup menu item **Hide Monitor**. To return to the monitor window, execute the configuration window's popup menu item **Show Monitor**.

- The copy configuration operation can be interrupted. When you copy a configuration, all objects that it contains are also copied. The time needed to completely copy a seemingly simple configuration can be deceiving, if the configuration has a very large containment

and/or reference network. You can halt the execution of the copy configuration operation by pressing the Ctrl and Backslash keys together.

When you halt the copy configuration operation, a dialog box is displayed prompting you about whether the operation should halt or continue. If you choose to halt the operation, the operation is aborted and the standard clean-up procedure is performed.

-
- i** **Tip:** For a detailed description of the features of the configuration copy, refer to the section “[Copying a Configuration](#)” on page 123. For information about specifying your session setup values, refer to the section “[Change Your Session Setup](#)” on page 224.
-

Performing the Configuration Copy	264
Getting Information about a Copy	265

Performing the Configuration Copy

Copy a configuration.

Procedure

1. Choose **Global Operations > Lock Configuration** from the configuration window's popup menu to lock the configuration.

Although locking the configuration is an optional step, it is recommended because it guarantees that the configuration entries are not modified or deleted by someone else while you perform the copy.
2. Choose **Build** from the configuration window's popup menu to update the display to exactly reflect the contents of the configuration. Although, building the configuration is also an optional step, it is recommended because someone may have deleted or moved an important entry before you started this configuration copy procedure. If you locked the configuration in step one, as you build it, all secondary entries are locked as they are added to the configuration.

When you lock and build the configuration, as recommended in steps 1 and 2, the configuration is *consistent*. You can check if a configuration is consistent, as well as view the status of other items, by choosing **Report > Configuration Info** from the configuration window's popup menu.

3. Choose **Global Operations > Copy** from the configuration window popup menu, which displays the Copy Configuration dialog box.
4. Specify the **Destination Directory**, **Replace Conflicting Objects**, **Update All References**, and **Preview** options.
 - **Destination Directory** Enter the directory where you want each configuration entry copied. If the destination directory that you specify does not exist, when you execute

the Copy Configuration dialog box, another dialog box appears, asking if you would like to create the directory.

- **Replace Conflicting Objects** Choose either to cancel the copy operation if the Pyxis Project Manager finds naming conflicts or to overwrite existing design objects with the same name.
- **Update All References**
- **Preview Only?** Choose to preview or not to preview the copy before it occurs, to allow you to confirm that the copy operation is correct.
 - i. If you choose **Yes**, when you execute the dialog box, a report window appears, displaying the *from* and *to* paths of each element of the configuration, and any naming conflicts between existing and copied objects. Only the report is generated; no copy occurs.
 - ii. If you choose **No**, when you execute the dialog box, the Copy Configuration command attempts to copy the configuration entries.
- **On Copy Error** If the copy process runs into errors, this option allows you to select whether to abort and cleanup (i.e. discard the copied information up to that point), or whether to continue the copy operation despite the errors.

5. Click **OK**.

Related Topics

[Change Your Session Setup](#)

[Get Information About a Design Configuration](#)

Getting Information about a Copy

If you save the configuration prior to copying it, the configuration object that records the configuration is automatically added to the configuration.

As a result, during a copy, the configuration object that records the configuration is copied along with the other configuration entries. If the configuration object is copied, you can get information about the copy in the destination directory.

To keep track of which entries are in the configuration, the configuration object has a reference to each configuration entry. During a copy, the Copy Configuration command places properties on the references from the copied configuration object to each of the copied entries. These properties record the source name and version of each copied configuration entry.

Procedure

1. Navigate into the directory to which you copied the configuration.

2. Open the configuration object that represents the configuration you copied by executing the popup item **Open > Configuration Window**, or execute **Windows > Open Configuration** from the pulldown menu.
3. Choose **Report > Entry Info** from the configuration's popup or pulldown menu.

A configuration monitor window appears and displays, among other information, the properties that the selected reference holds. Among the other properties are these two:

- a. A property named **from_path**. The value is the pathname of the configuration entry from which this copy was made.
- b. A property named **from_version**. The value is the version number of the configuration entry from which this copy was made, indicating the version number at the time of the copy. The source design object's version number changes as the design object evolves.

Related Topics

[\\$copy_configuration\(\)](#)

[\\$report_entry_info\(\)](#)

Releasing a Design Configuration

Releasing a configuration creates a *protected* copy of its entries.

In other words, the Pyxis Project Manager does not allow you to create new versions of these released design objects. A copy becomes protected when the Release Configuration command assigns the “released” attribute to every versioned object in the release directory. If you want to open and to evolve these design objects, you must copy the released configuration to another directory.



Note

The time needed to completely release a seemingly simple configuration can be deceiving, if the objects in the configuration have a very large containment and/or reference network. You can interrupt the release operation by pressing the Ctrl and Backslash keys together.

A major difference between release and copy is that release does not always overwrite conflicting design objects in the destination directory. Instead, for versioned design objects only, release *merges* the conflicting object with the one being copied, making the copy the most current version of the design object in the destination directory. When the release operation merges the objects, the new version is overlaid on top of the older version, and given the next version number available. In the case of conflicts with unversioned objects, the unversioned object is overwritten.

Caution



Be aware that release automatically overwrites previously released unversioned design objects in the destination directory. This rule also applies to unversioned containers, except for objects of the type “mgc_container,” which are ordinary directories. Unversioned containers not only overwrite the container object, but all of the container's contents. For this reason, most containers, other than those of type “mgc_container”, are defined as versioned to allow layering to occur during a release.

Releasing a configuration releases a single version of each design object. The first time that you release design objects to the destination directory, all versioned design objects are rolled to version 1. In subsequent releases to the same destination directory, new versions are layered on top of the existing versions, creating a version history of the released configuration. Tracking of library component versions is not present in the release.

Note



Tracking of released library component versions is not maintained. All design object versions are initially released to the destination directory as version 1, and subsequent versions are layered and numbered sequentially.

The “released” attribute only prevents you from opening and evolving versioned objects in the release directory. It does not prevent you from opening an unversioned design object, or from deleting or moving any released design object. To preserve the integrity of your release, avoid opening, deleting, or moving design objects in the release directory.

For more information about release concepts, refer to the section “[Releasing a Configuration](#)” on page 125.

Performing the Release	267
Getting Information about a Release	269

Performing the Release

Release a design configuration.

Procedure

1. If the configuration is untitled, save it by choosing **Save As** from the configuration window's popup menu or **File > Save As** from the pulldown menu, and then specifying a pathname for the configuration object in the prompt bar, which is displayed.

Saving the configuration automatically causes the configuration to include the configuration object. Inclusion of the configuration object is important for two reasons:

- With the configuration object part of the configuration, you can recreate the configuration at a later time.

- As with the Copy Configuration command, the Release Configuration command stores important record keeping information in properties on the references between the configuration object and the configuration entries.

If the configuration object is not included in the release, this record keeping information is not kept.

You can open a released configuration object, but as with other released design objects, you cannot create a new version. In addition, the resulting configuration window greys out any operations that could change the configuration.

2. Choose **Global Operations > Lock Configuration** from the configuration window's popup menu to lock the configuration.

Although locking the configuration is optional, it is recommended because it guarantees that the configuration entries are not modified or deleted by someone else while you perform the release.

3. Choose **Build** from the configuration window's popup menu to update the display to exactly reflect the contents of the configuration.

Although building the configuration is optional, it is recommended because someone may have deleted or moved an important entry before you started this procedure. If you locked the configuration in step one, as you build it, all secondary entries are locked as they are added to the configuration.

As the build command executes, status information is reported in the monitor window. If your session setup values specify to show the monitor window, the monitor window is opened when you execute this function. When the operation completes, the monitor window remains visible until you remove it by executing the monitor window's popup menu item **Hide Monitor**. Until you execute another configuration operation, you can revisit the build command's status report, by executing the configuration window's popup menu item **Show Monitor**.

When you lock and subsequently build the configuration, as in steps 1 and 2, the configuration is *consistent*. You can check if a configuration is consistent and view the status of other items by executing **Report > Configuration Info** from the configuration window's popup or pulldown menu.

4. Choose **Global Operations > Release** from the configuration window popup menu, which displays the Release Configuration dialog box.
5. In the Destination Directory field, enter the directory where you want each configuration entry to be released.
6. In the Annotation field, enter an annotation for the release.

You have the option of leaving this prompt blank. During the release, the value that you specify is added as a version property to the version of the configuration object created at the release directory. If the configuration is untitled, your annotation is not saved.

7. Select a **Preview Report Only?** button.

You can preview the release before it occurs, to confirm that the release operation is correct.

- If you choose **No**, when you execute the dialog box, the Release Configuration command attempts to release the configuration.
- If you choose **Yes**, when you execute the dialog box, an information window appears and reports the *from* and *to* paths of each element of the configuration and any naming conflicts between existing and released objects. Only the report is generated; no release occurs. If you specify a destination directory that does not exist, the Pyxis Project Manager creates it during the preview and then removes it after the operation is complete.

If your session setup values specify to show the monitor window, the monitor window is opened when you execute this function. The monitor window remains visible until you return to the configuration window, by executing **Hide Monitor** from the monitor window's popup menu. You can return to the monitor window by executing the configuration's popup menu item **Show Monitor**.

8. Specify whether to update the references of the released configuration by clicking on the desired button.

- If you click on the **No** button, the references of the released design objects continue to point to the original destination of their targeted objects. This is a useful option when using a location map. You can update the location map instead of updating the references of the design objects.
- If you click on the **Yes** button, the references of the released design objects point to their targeted objects at their new destination.

9. Click **OK**.

Related Topics

[Getting Information about a Release](#)

[\\$hide_monitor\(\)](#)

[Change Your Session Setup](#)

[\\$show_monitor\(\)](#)

[Get Information About a Design Configuration](#)

Getting Information about a Release

If you save the configuration prior to releasing it, the configuration object that records the configuration is automatically added to the configuration.

As a result, during a release, it is copied along with the other configuration entries. If the configuration object is in the release, you can get information about the release in the

destination directory. The Release Configuration command stores the following types of information during a release:

- Information about the design objects that were the source of the release. This information is stored as properties on the references between the source configuration object and each source configuration entry. It includes the path to which the design object was released, who released it, and the date of the release.
- Information about the released design objects. This information is further divided into two groups:
 - a. Information stored as properties on the references between the released configuration object and each released configuration entry. The information includes the pathname of the design object of which this is a protected copy, who released it, and the date of the release.
 - b. The release annotation. This information is the text string that you entered in the Release Configuration dialog box. It is stored as a version property of the released configuration object.

Procedure

1. Navigate into the directory that contains the source configuration object.
2. Select the source configuration object and open it by executing the popup menu item **Open > Configuration Window**. Alternately, execute **Windows > Open Configuration** from the pulldown menu.
3. Choose **Report > Entry Info** from the configuration's popup or pulldown menu.

A configuration monitor window displays the properties that the selected reference holds. Included in these properties are the following:

- **released_to_path**—the value is the pathname to which this design object was released.
- **released_by**—the value is the user login name of the person who last released design object.
- **release_date**—the value is a UNIX time integer value that indicates when this design object was released.
- **release_date_string**—the value is a text string that indicates when this design object was released.

To view information about the individual released design objects, perform the following steps:

1. Navigate into the directory to which you released the configuration.
2. Select the configuration object that represents the configuration that you released and open it by executing the popup menu item **Open > Configuration Window**.

3. Choose **Report > Entry Info** from the configuration window's popup menu.

A configuration monitor window displays the properties that the selected reference holds. Included in these properties are the following:

- **from_path** — the value is the pathname of the design object of which this is a protected copy.
- **from_version** — the value is the version number of the design object of which this is a protected copy. This property indicates the version number at the time of the release. The source design object's version number changes as the design object evolves.
- **released_by** — the value is the user login name of the person who released this design object.
- **release_date** — the value is a UNIX time integer value that indicates when this design object was released into this directory.
- **release_date_string** — the value is a text string that indicates when this design object was released into this directory.

To view the annotation that you created when you released the configuration, perform the following steps:

1. Navigate to the directory to which you released the configuration.
2. Select the configuration object that represents the released configuration.
3. Choose **Report > Show Versions** from the navigator popup or pulldown menu.

A version window appears, displaying the configuration object's versions.

4. Select the desired version.

To view the annotation of the latest release into this directory, select the highest numbered version.

5. Choose **Report Version Info** from the version window's popup or pulldown menu.

A read-only window appears, displaying information about the selected version. In the information window, look for the Version Properties heading. The version's properties are beneath that heading. For example, one version property is named **release_comments**, and its value, the annotation, is displayed in quotes, to the right.

Related Topics

[\\$report_version_info\(\)](#)

Changing the References of a Design Configuration

You can change the references of the entries contained in your configuration.

Changing references in a configuration window works similarly to changing references in a navigator. The only difference is the set of objects whose references you can change. That is, using a navigator, you can select several objects and change their references. Those several objects become the set to which you apply the Change Object References command. With a configuration, all the configuration's entries are the set to which you apply the Change Configuration References command.

Changing a configuration's references requires that you supply a pathname against which the selected objects are compared. The portion of the reference path that matches the pathname is substituted with a string that you supply. The Change Configuration References command does not check to see if the resulting reference points to an existing object.

Procedure

1. Choose **Global Operations > Change Reference** from the configuration window's popup menu.

A dialog box appears, prompting you to enter the current reference Pattern the new reference Replacement, and to specify whether you want to preview the results of the operation.

2. In the Pattern field, enter the pathname of the reference you want to change.

You can specify the Pattern field using UNIX System V wild cards.

3. In the Replacement field, enter the new pathname.

The Pyxis Project Manager matches existing references with the pathname you entered in the Pattern field. When a match occurs, the Pyxis Project Manager substitutes the pathname you entered in the Replacement field.

4. If you want to preview the expected results of this operation, before actually performing the operation, select **Yes** in the Preview Only field; otherwise, select **No**.

- If you select **Yes**, the expected results of the operation, including the *from* and *to* paths of each entry's changed references, are written to the monitor window. Only the report is generated; no references are changed.
- If you select **No**, the operation is performed and the results, including the *from* and *to* paths of each entry's changed references, are written to the monitor window.

The status of the change configuration references operation is reported and displayed in real-time to the monitor window. If your session setup values specify to show the monitor window, the monitor window is opened when you execute this function. The monitor window remains visible until you execute the monitor window's popup menu item **Hide Monitor**.

Note



The safest way to change the references of your configuration is to first execute the `$change_configuration_references()` function with the `@preview` switch. This switch causes the function to report the expected results of the function execution to the monitor window, without actually making any changes. If the report accurately reflects your intended actions, you can execute the function with the `@nopreview` switch and the changes are made.

You can interrupt the Change Configuration References command at any time by pressing the Ctrl and Backslash keys together. When you halt the change configuration references operation, a dialog box is displayed prompting you about whether the operation should halt or continue. If you choose to halt the operation, the operation is aborted and the standard clean-up procedure is performed.

Related Topics

[Changing Reference Pathnames](#)

[Regular Expressions](#)

[Change Your Session Setup](#)

[\\$change_configuration_references\(\)](#)

Freezing and Unfreezing a Design Configuration

You can prevent the Pyxis Project Manager version depth mechanism from automatically deleting versions beyond the version depth setting by freezing the configuration.

Remember that the entries in a configuration are versions of objects. Freezing a configuration freezes all these versions. Unfreezing reverses the freeze and allows the version depth mechanism to delete any version that exceeds the version depth.

If you *unfreeze* a version that is older than the version depth, that version is not immediately pruned, but survives until you either set the version depth or create a new version of the design object. You can only freeze and unfreeze versioned objects. Although freezing a version prevents you and the automatic version depth mechanism from deleting that version, it does not prevent you from deleting an entire design object that has a frozen version.

Procedure

- To freeze a configuration, choose **Global Operations > Freeze** from the configuration window's popup menu.
- To unfreeze a configuration, choose **Global Operations > Unfreeze** from the configuration window's popup menu.

Related Topics

[\\$freeze_configuration\(\)](#)

Lock a Design Configuration

Locking your configuration is not a requirement; you can release and copy configurations without locking them.

However, after you add primary entries, it is recommended that you lock your configuration before releasing it. Usually, when you release your configuration, you first lock it, then build it, and finally, release it.

Locking your configuration before release prevents anyone from changing any of the objects that make up the configuration. Locking guarantees that the entries in the release are consistent with the actual data stored on the disk.

Read-only and Write Locking Modes.....	274
Locking a Design Configuration.....	274

Read-only and Write Locking Modes

The Pyxis Project Manager gives you a choice of two locking modes: read-only and write.

A *read-only* lock prevents others from changing or deleting objects entered in the configuration, while allowing others to 'read' the locked entries. Read-only lock privileges include copying, viewing, or showing information; a read-only lock is also known as a shared lock. In most cases, the only locking mode you use is read-only.

A write lock prevents all others (except for you) from editing the locked objects. A write lock is also known as an exclusive lock, because only you can change the locked object. You may want to use a write lock if you intend on using the Change Configuration References command. In this way, you can change object references while preventing others from making changes during the process.

Related Topics

[Locking a Design Configuration](#)

Locking a Design Configuration

To lock a configuration, you must have write permission for the directory that contains the configuration objects.

Procedure

1. Choose **Global Operations > Lock Configuration** from the configuration window's popup menu.
A dialog box appears, allowing you to select a **Read-Only** or **Write** lock mode.
2. Click on **Read-only** or **Write** locking mode.

3. Click **OK**.

Related Topics

[\\$lock_configuration\(\)](#)

[\\$unlock_configuration\(\)](#)

Deleting a Design Configuration

After you build a configuration, you can delete it. Deleting a configuration deletes the design object versions that are represented in the configuration.

Deleting each version represented in the configuration does not delete the corresponding design object; rather, it deletes only the specific version that appears in the configuration. For example, if design object *fred* has versions 3, 5, and 8, and if the configuration includes only version 5, deleting the configuration leaves *fred* with versions 3 and 8.

Caution



Be aware that because unversioned design objects have only one version, deleting a configuration that contains unversioned design objects deletes these objects entirely.

Procedure

1. Choose **Global Operations > Delete Configuration** from the configuration window's popup menu.

The delete configuration command reports real-time status to the monitor window, throughout the execution of the operation.

If your session setup values specify to show the monitor window, the monitor window is opened when you execute this function.

2. To return to the configuration window, execute the monitor window's popup menu item **Hide Monitor**. You can return to the monitor window by executing the configuration's popup menu item **Show Monitor**.
3. At any time during the execution of the command, you can press the Ctrl and Backslash keys together to interrupt the operation. Deleting a configuration, using the Delete Configuration command, is an interruptible operation. When you halt the delete configuration operation, a dialog box is displayed prompting you about whether the operation should abort or continue. If you choose to abort the operation, the operation is halted.

Interrupting the delete configuration command does not return the design configuration to its original state. Any objects in the configuration that have been deleted prior to the interrupt are permanently deleted; all others are not deleted.



Tip: For information about specifying your session setup values, refer to the section “[Change Your Session Setup](#)” on page 224.

Related Topics

[Change Your Session Setup](#)

[\\$delete_configuration\(\)](#)

Archive and Restore Designs

After you release your configuration, you may want to archive the configuration in its current form for later use. For this purpose, the Pyxis Project Manager provides you with an archiving and restore facility.

To archive a configuration, you must use of the same platform environment for both the configuration's source and its destination. That is, the platform type and operating system version are compatible, the Mentor Graphics Tree version and tools version are the same, and the required type registries and fonts are the same. Various parts of the design may be archived separately; however, this may result in much more work when you begin updating references in “[Restoring a Configuration](#).”

Archiving a Configuration	276
Restoring a Configuration.....	277
Updating Archive References.....	277
Checking References After Archiving	278

Archiving a Configuration

Archive your configuration in its current form for later use.

Procedure

1. Create a new configuration.
2. Save the configuration, by executing the session window's pulldown menu item **File > Save As** or **Save**. Enter the path in the prompt bar.
3. Copy or release the configuration to an empty directory by using one of the navigator popup menu items **Edit > Copy** or **Edit > Release**.
4. Archive the directory from the previous step, using the UNIX commands tar, cpio, or rbak.

Related Topics

[\\$\\$create_configuration\(\)](#)

[Restoring a Configuration](#)

Restoring a Configuration

Restore your configuration.

Procedure

1. Restore the archived directory, using the tar, cpio, or rbak commands.
2. Note the pathname of the archived configuration in its new location.
3. Update the references within the archived configuration which point to other objects within the configuration, to the new pathname you noted in step 2.
4. Update the references held by objects in the archived configuration, which point to objects not in the configuration just restored.

Related Topics

[Updating Archive References](#)

[Checking for Broken References](#)

Updating Archive References

To update an archived configuration's references, replace the source directory pathname (the directory pathname from which the archive was created) with the target directory pathname (the directory pathname to which the configuration is being restored).

Procedure

1. Activate a navigator.
2. Select your design object `$PROJ_DESIGN/brads/data/archive_cfg`
3. Type the following command line and press the Return key:

```
$change_object_references ("$PROJ_DESIGN/brads/data/",  
" $ARCHIVE_DESIGN/workstation7/re_store_area/data/", @all,  
@nopreview);
```

The `@all` switch indicates that the references of all versions of the selected design object are changed. The `@nopreview` switch indicates that this command's execution is *not* a preview of the actual execution. Therefore, the example above changes the references of all the versions of the selected design object whose pathname begins with the string `“$PROJ_DESIGN/brads/data/”` to references with pathnames that begin with the string `“$ARCHIVE_DESIGN/workstation7/restore_area/data”`.

Note



One way to safely change the references of your design object is to first execute the `$change_object_references()` function with the `@preview` switch. This switch causes the function to report the expected results of the function execution to the monitor window, without actually making any changes. If the report accurately reflects your intended actions, you can execute the function with the `@nopreview` switch and the changes are made.

Another way to safely change the references of your design is to experiment by changing the references of a single design object, before changing the references of all the objects.

To do this, make a copy of the `archive_cfg` object and change its references. To check the copy's references, you can execute the **Check Broken References** command from the popup command line or from the navigator's popup or pulldown menu item **Edit > Check Refs.**

Examples

For a demonstration of this replacement process, assume the original source directory pathname of the configuration `archive_cfg` is as follows:

`$PROJ_DESIGN/brads/data`

This pathname is replaced with the path of the new target directory, as shown:

`$ARCHIVE_DESIGN/workstation7/restore_archive/data`

In this example, the references of the restored configuration have all pathnames with initial strings of `$PROJ_DESIGN/brads/data` replaced with the pathname of the target directory `$ARCHIVE_DESIGN/workstation7/restore_archive/data`. Therefore, `$PROJ_DESIGN/brads/data/archive_cfg` becomes `$ARCHIVE_DESIGN/workstation7/restore_archive/data/archive_cfg`.

You use the **Change/ Fix References** command to update references from the navigator. The **Change Object References** command should be performed on the target directory from the navigator in a popup command line.

Related Topics

[`\$change_object_references\(\)`](#)

Checking References After Archiving

Troubleshoot broken references.

Procedure

1. Select your design object *archive_cfg*.
2. Type the following command line and press the Return key:

```
$check_references(@notraverse, 10 );
```

This command reports all the broken references in the design object. By specifying the @notraverse option, you exclude all externally referenced objects.

Fix or redirect the broken references to your design object from the Fix Broken References dialog box as follows:

- a. Select the broken reference pathname that you want to fix.
- b. In the From: field, enter the broken reference pathname.
- c. In the To: field, enter the correct reference pathname
\$ARCHIVE_DESIGN/workstation7/restore_archive/data/archive_cfg.
- d. Repeat steps a through c, until all your broken references are corrected.
- e. Execute the Fix Broken References dialog box by clicking the **OK** button.

Related Topics

[\\$change_object_references\(\)](#)

[\\$check_references\(\)](#)

Managing Designs Using iDM

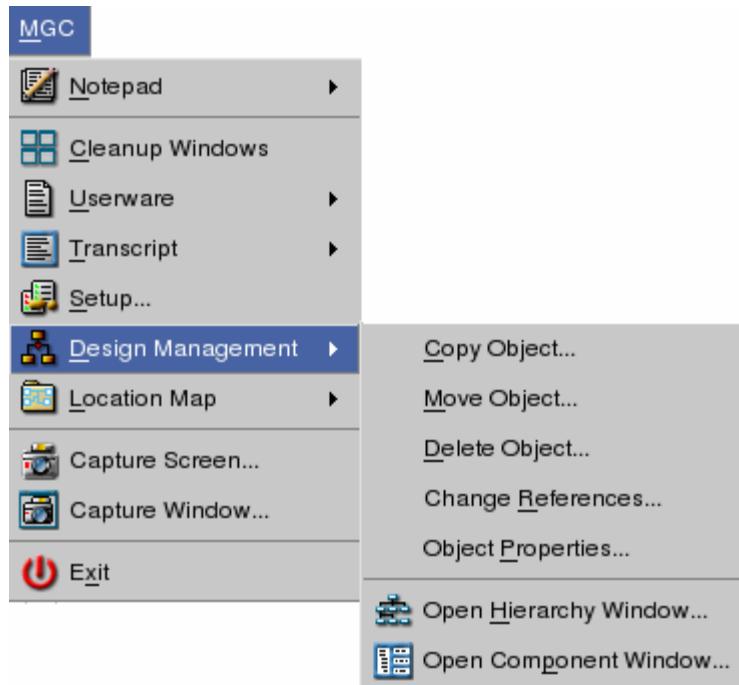
You can perform iDM design management operations including moving, copying, deleting, and changing the references of your design objects and viewing a component's design hierarchy.

The iDM commands can be accessed from the following methods:

- From the pulldown menu in any Pyxis Project Manager window: Execute the desired iDM task under the **MGC > Design Management** menu option.
- From the popup menu in the Session Window: Click the mouse (Menu) button or press the F4 function key, and execute the desired iDM task under the **MGC > Design Management** menu option.

Figure 3-44 shows the menu paths to the iDM commands.

Figure 3-44. MGC Design Management Menu



Copying Design Objects	280
Moving Design Objects	283
Deleting Design Objects	287
Changing References on Design Objects	288
Display a Component Hierarchy	290
Displaying Component Information	293

Copying Design Objects

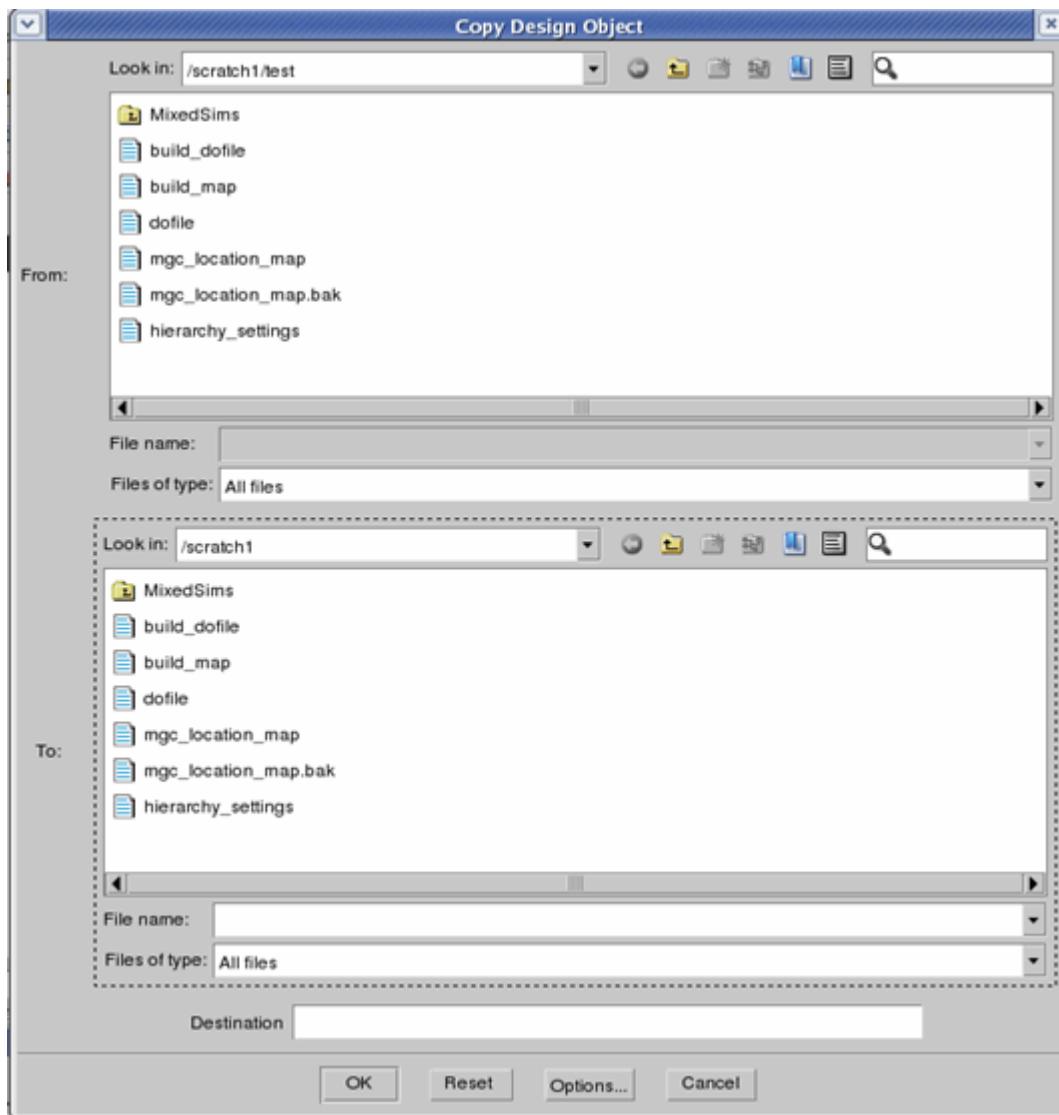
You can copy a single design object or multiple design objects using the iDM Copy Design Object command.

Procedure

1. Execute the **MGC > Design Management > Copy Object** pulldown menu item.

The Copy Design Object dialog box, as shown in Figure 3-45, appears, in which you specify the source design objects, the destination container, and the behavior of the copy operation.

Figure 3-45. Copy Design Object Dialog Box



2. In the top object browser dialog, select the source design objects that you want to copy.

You can press the Ctrl key as you click the mouse Select button to select multiple entries, or you can drag the mouse button over the list area to select multiple entries that are adjacent. You can also use the navigation buttons to explore objects and references or to move to a new directory.

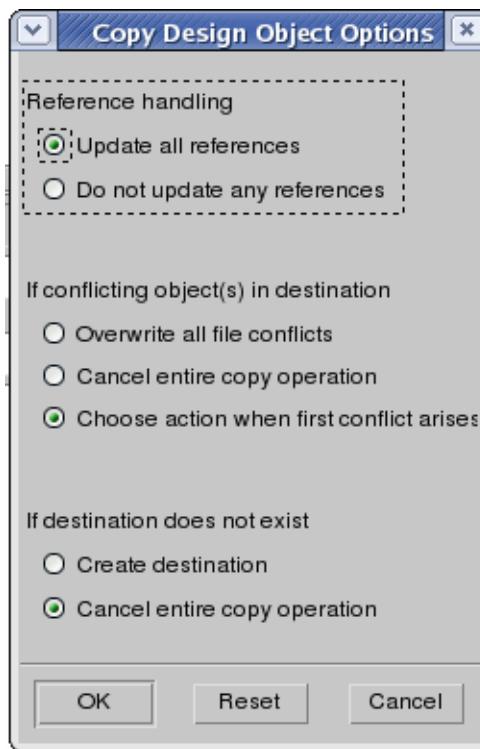
3. In the bottom object browser dialog, select the destination container for the copy operation.

You can only select a single destination container. When you select the destination, the absolute pathname of the destination is displayed in the Destination field below the right object browser dialog. You can also specify the destination container without using the object browser dialog, by typing either the absolute or relative pathname directly into

the Destination field. If you specify a relative pathname, the relative pathname is resolved by appending the relative pathname to the end of the pathname displayed at the top of the To Destination: object browser dialog. For example, if the relative pathname *base* was entered into the Destination field in Figure 3-45, the resolved pathname would be *\$PROJECT_XYZ/project2/base*.

4. Specify the behavior of the copy operation, by clicking the **Options** button on the Copy Design Object dialog box. The Copy Design Object Options dialog box appears, as shown in Figure 3-46.

Figure 3-46. Copy Design Object Options Dialog Box



5. The Copy Design Object Options dialog box allows you to specify the following:
 - Whether references between the copied design objects are updated to reflect their new location. By default, references are automatically updated.
 - Whether to replace conflicting design objects or cancel the copy operation. By default, the copy operation is canceled.
 - Whether to create a destination if the destination container does not exist or cancel the copy operation. By default, the copy operation is canceled.
6. Execute the Copy Design Object Options dialog box by clicking OK, after selecting your copy options.

The dialog box disappears, and you return to the Copy Design Object dialog box.

7. Execute the Copy Design Object dialog box by clicking OK.

Related Topics

[\\$copy_design_object\(\)](#)

Moving Design Objects

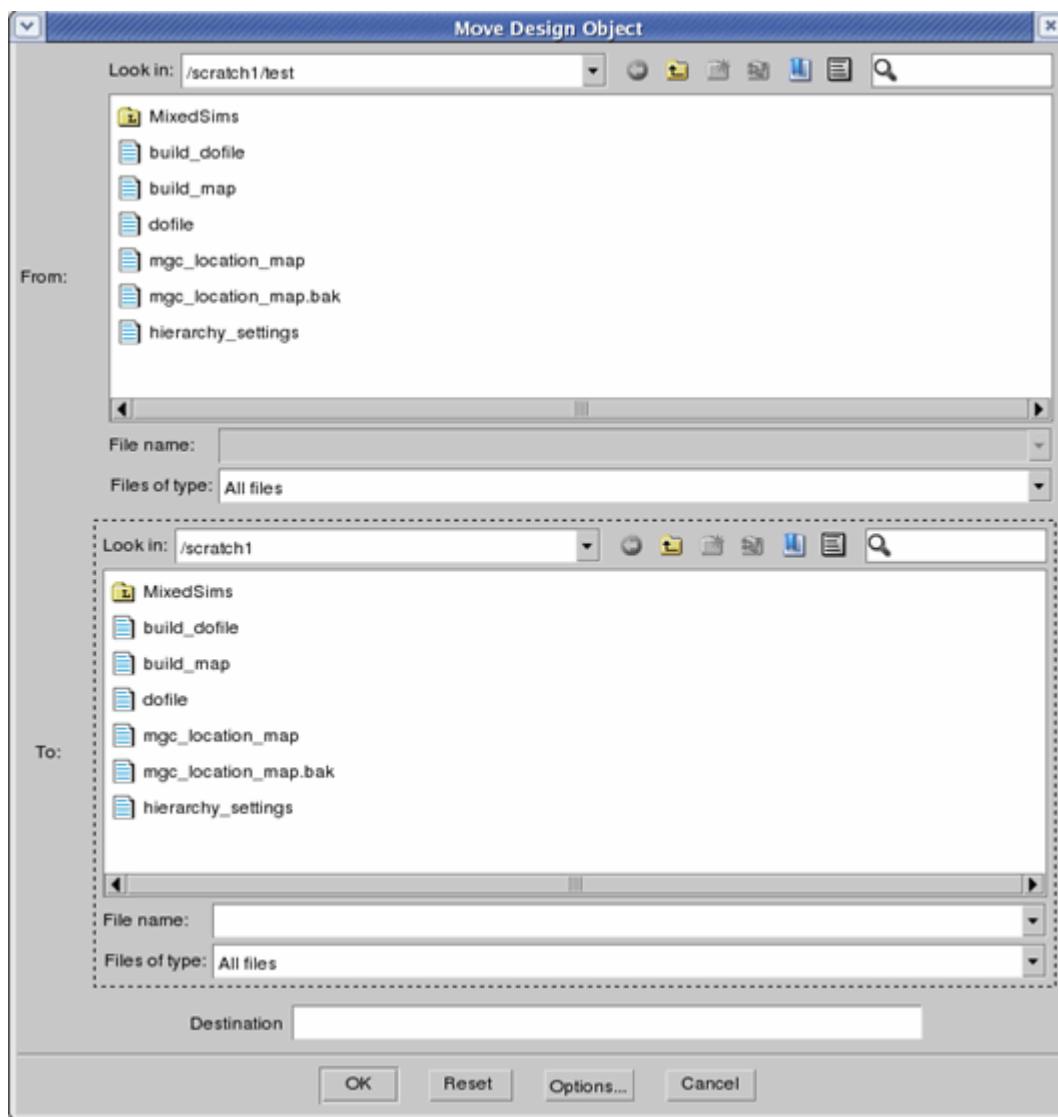
You can move a single design object or multiple design objects using the iDM Move Design Object command.

Procedure

1. Execute the **MGC > Design Management > Move Object** pulldown menu item.

The Move Design Object dialog box, shown in Figure 3-47 appears, in which you specify the source design objects, the destination container, and the behavior of the move operation.

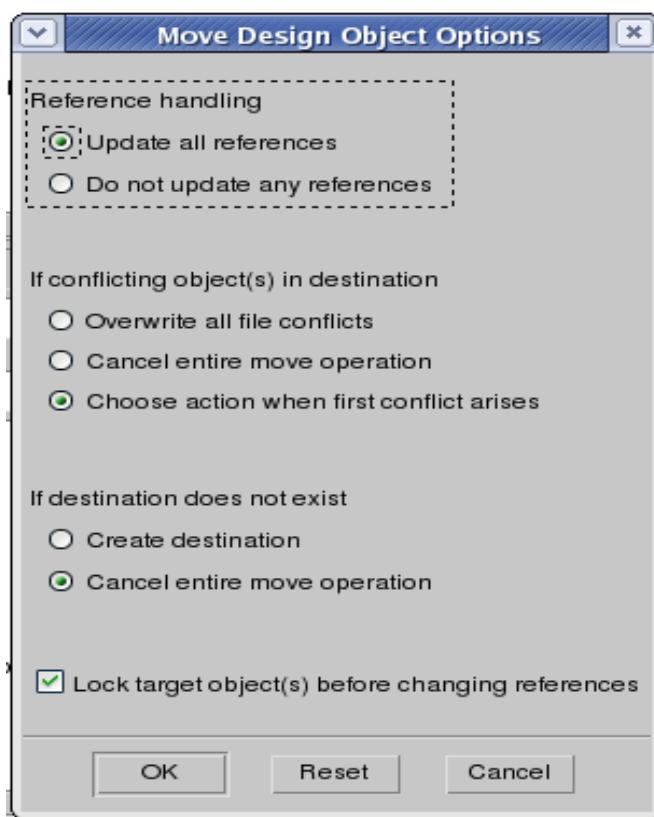
Figure 3-47. Move Design Object Dialog Box



2. In the top object browser dialog (the “From” pane): Select the source design objects that you want to move. You can press the Ctrl key as you click the mouse Select button to select multiple entries, or you can drag the mouse button over the list area to select multiple entries that are adjacent. You can also use the navigation buttons to explore objects and references or to move to a new directory
3. In the bottom object browser dialog (the “To” pane): Select the destination container for the move operation. You can only select a single destination container. When you select the destination, the absolute pathname of the destination is displayed in the Destination field below the right object browser dialog.
4. Type the **File name** of the new file that contains the moved object.

5. You can also specify the destination container without using the object browser dialog, by typing either the absolute or relative pathname directly into the Destination field. If you specify a relative pathname, the relative pathname is resolved by appending the relative pathname to the end of the pathname displayed at the top of the To Destination: object browser dialog. For example, if the relative pathname *base* was entered into the Destination field in Figure 3-47, the resolved pathname would be *\$PROJECT_XYZ/project2/base*.
6. To filter the overwrite options and to specify the behavior of the move operation, click the **Options** button on the Move Design Object dialog box. This brings up the Move Design Object Options dialog box, as shown in Figure 3-48.

Figure 3-48. Move Design Object Options Dialog Box



7. The options in Figure 3-48 have the following meaning:

- **Reference Handling**

This specifies whether references between the moved design objects are updated to reflect their new location. By default, references are automatically updated. Select any one of these two options to handle any references that are bound to the design object that is being moved:

- **Update all references** option: All references are updated to reflect the location of the moved object. By default, this option is checked.

- **Do not update any references** option: The references are not updated for the design object after it is moved to the new location.
- **If conflicting object(s) in destination**

With this option, you specify the action that the Move Design Object takes when a design object already exists in the destination with the same name as the source design object, and whether to replace conflicting design objects or cancel the move operation.

When a design object exists in the destination container with the same name as a design object being moved, a prompt is displayed with options to overwrite/ cancel the move. The conflicting design object in the destination container is replaced with the source design object if the conflicting design object is not a container. To ensure the integrity of your data, container design objects are never overwritten.

If the design object is a directory, then the move operation of conflicting directories is not allowed.

- **Overwrite all file conflicts** option — Replaces conflicting non-container type design objects with copies of the selected design objects that were moved.
- **Cancel entire move operation** option — Cancels the entire copy operation if any name conflicts are found at the destination.
- **Choose action when first conflict arises** — For each selected design object, if a naming conflict is found, the Move Object command asks if you want to leave the destination object intact or to replace it with the moved one. If an overwrite conflict occurs and an action is not specified, this is the default.

- **If destination does not exist**

If the destination specified for the move operation does not exist, any of the following two options may be selected:

- **Create destination** option — A new destination is created, as specified in the Destination field of the Move Design Object dialog box, and the object is moved there.
- **Cancel entire move operation** option — Cancels the entire move operation if the destination path does not already exist. By default, the move operation is canceled.

- **Lock target object(s) before changing references**

With this checkbox checked, the target design object is locked before attempting to change its references to point to its new location. By default, the design objects are locked before their references are updated.

Although not checking this option allows the move operation to execute faster, you should only use this option when you are absolutely sure that the target design

objects are not changed by another user during its execution. If the source design object is modified while this operation is executing, your data can be corrupted.

8. Click OK to execute the Move Design Object Options dialog box. The dialog box disappears, and you return to the Move Design Object dialog box.
9. Click OK to execute the Move Design Object dialog box.

Related Topics

[\\$move_object\(\)](#)

Deleting Design Objects

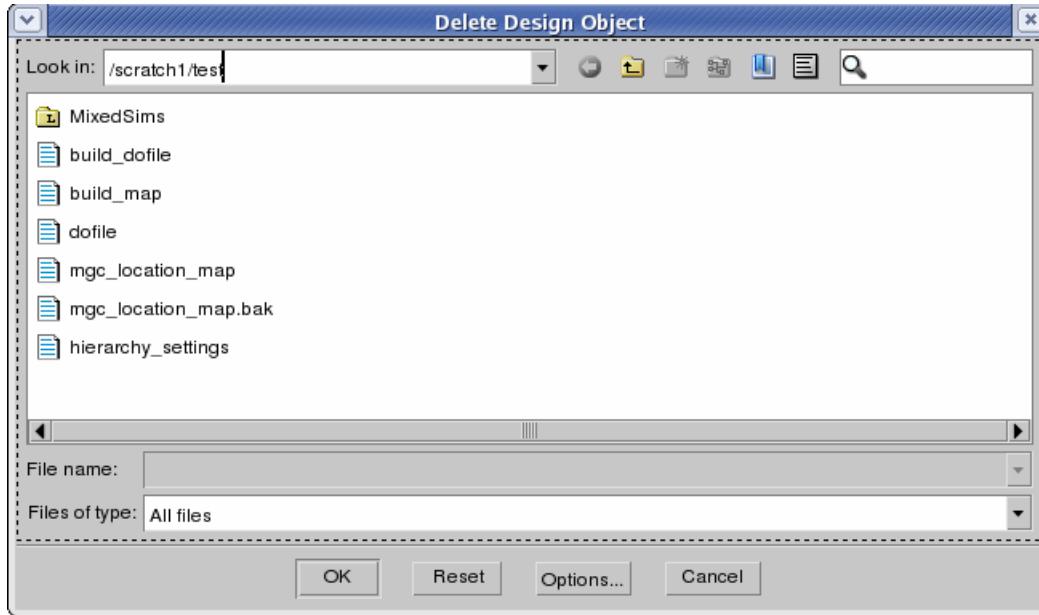
You can delete a single design object or multiple design objects when you use the iDM Delete Design Object command.

Procedure

1. Execute the **MGC > Design Management > Delete Object** menu item.

The dialog box, shown in Figure 3-49, appears.

Figure 3-49. Delete Design Object Dialog Box



2. Select the design objects that you want to delete by clicking the mouse Select button in the object browser dialog list area.

You can press the Ctrl key as you click the mouse Select button to select multiple entries, or you can drag the mouse button over the list area to select multiple entries that

are adjacent. You can also use the navigation buttons to explore objects and references or to move to a new directory.

3. Click on the **Options** button to display the Delete Design Object Options dialog box which is shown in Figure 3-50.

Figure 3-50. Delete Design Object Options Dialog Box



4. Specify the behavior of the delete operation by clicking one of the buttons in the field entitled "If A Container Has Locked Objects".

This field indicates whether the delete operation should delete a container design object that contains a locked design object. By default, design objects that contain locked design objects are not deleted.

5. Execute the Delete Design Object Options dialog box by clicking OK.

The Delete Design Object dialog box is still displayed.

6. Execute the Delete Design Object dialog box by clicking OK.

The selected design objects are permanently deleted.

Related Topics

[\\$delete_object\(\)](#)

Changing References on Design Objects

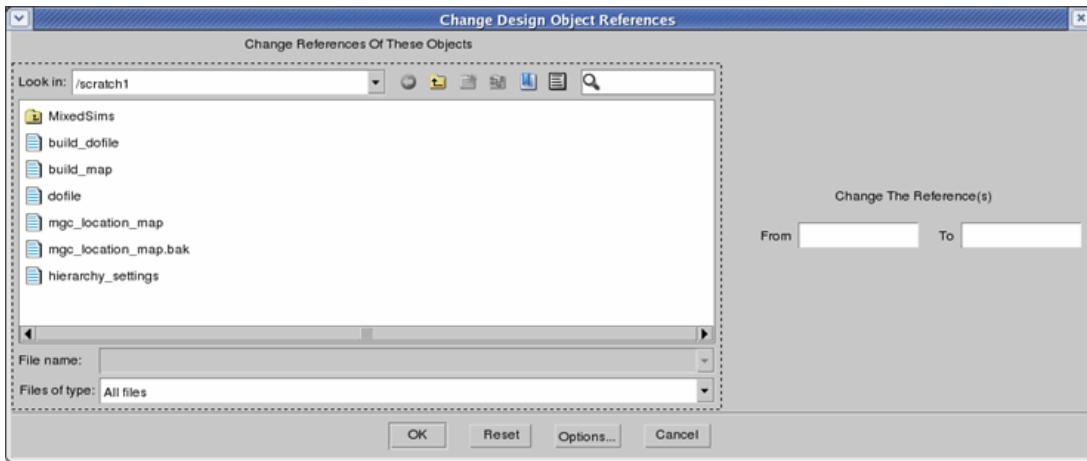
You can change the references of all versions of either a single design object or multiple design objects using the iDM Change Design Object References command.

Procedure

1. Execute the **MGC > Design Management > Change References** pulldown menu item.

The Change Design Object References dialog box shown in Figure 3-51 appears, in which you specify the design objects whose references you want to change and the behavior of the change references operation.

Figure 3-51. Change Design Object References Dialog Box



2. Select the design objects whose references you want to change by clicking the mouse Select button in the object browser dialog list area.

You can press the Ctrl key as you click the mouse Select button to select multiple entries, or you can drag the mouse button over the list area to select multiple entries that are adjacent to each other. You can also use the navigation buttons located on the right of the list area to explore objects and references or to move to a new directory.

3. In the From field, enter the portions of the reference pathnames that you want to change.

Each time you specify a string in the From field, an additional From and To field is displayed. This feature allows you to enter multiple From and To strings. The Change Design Object References command allows you to specify the From pattern strings using System V regular expressions.

4. In the To field, enter the text pattern that you want substituted in place of the From field text string.

When you specify a replacement pattern, the change references operation substitutes the replacement pattern into the reference exactly as you typed it. Hard pathname patterns are not first converted to soft pathnames before being substituted. Therefore, to ensure the integrity of your data, you should enter soft pathnames into the To field whenever possible.

5. Specify the behavior of the change references operation by clicking the dialog box's **Options** button, which displays the Change Design Object Reference Options dialog box.

6. In the Change Design Object Reference Options dialog box, specify the behavior of the change references operation by clicking the Lock Source Object(s) field.

The Lock Source Object(s) field indicates whether the change references operation should lock each source design object before changing its references. By default, a design object is locked before its references are changed.

7. Execute the Change Design Object Reference Options dialog box by clicking OK.

The Change Design Object Reference Options dialog box disappears, and the Change Design Object dialog box is still displayed on your screen.

8. After you have checked that you have specified the correct design objects in the object browser dialog list area and the correct To and From string patterns in the Change the Reference(s) field, execute the Change Design Object References dialog box by clicking OK.

In each of the references of the specified design objects, the iDM Change Design Object References command attempts to match each of the specified pathname strings until a successful match is found. When a successful match is found, the change references command replaces the matched pattern with the replacement string and then proceeds to the next reference. If a reference contains more than one matching string, only the first match is changed.

Related Topics

[\\$change_design_object_references\(\)](#)

[Regular Expressions](#)

Display a Component Hierarchy

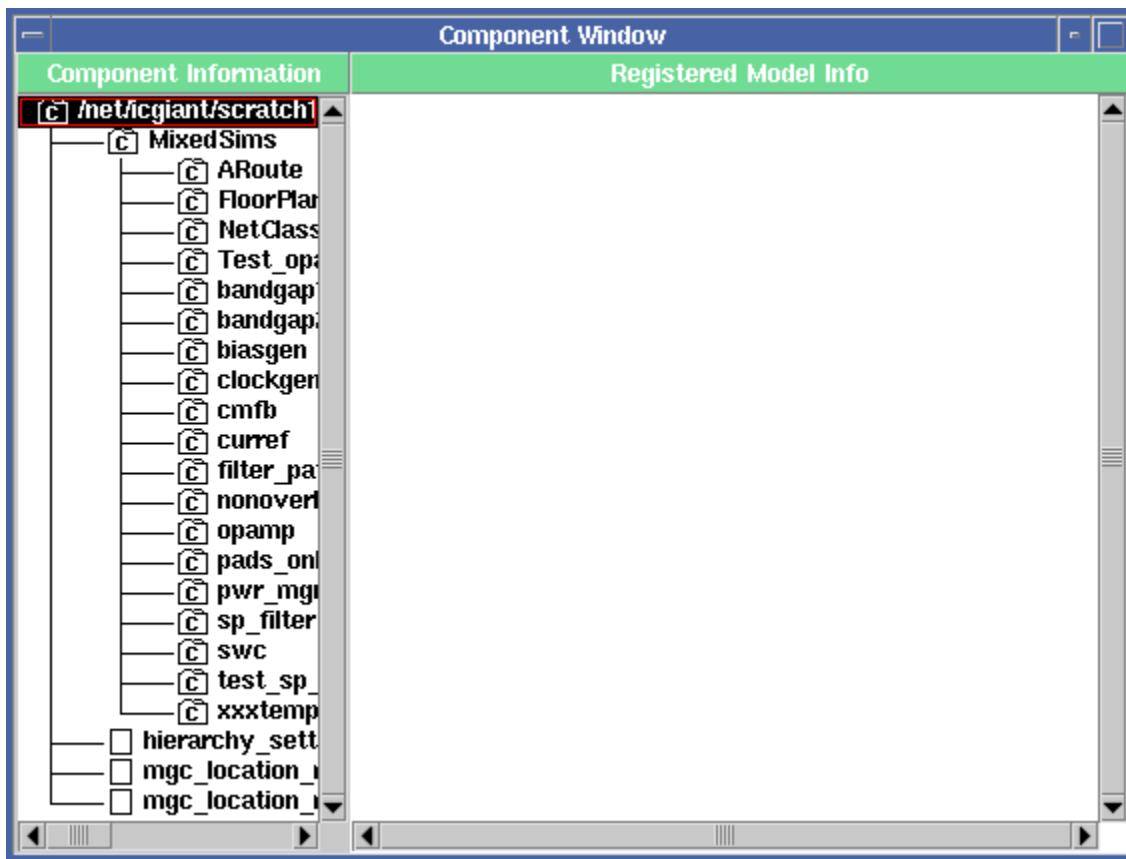
You can display your component hierarchy by using the iDM Hierarchy Window.

The hierarchy displayed is a physical or logical hierarchy as it is defined by the component level of the design. The Hierarchy Window displays this information graphically so that you can view directly, information concerning the instance names, property values, object designations and model information.

You use the **MGC > Design Management > Open Hierarchy Window** from the pull down menu of your application whenever you want to view a particular hierarchy. Because the Hierarchy Window is customized to be best utilized for the application from which it is displayed, there are different incarnations that are specific to that application. For example:

- The Component Hierarchy Window displays as in Figure 3-52.

Figure 3-52. Component Hierarchy Window



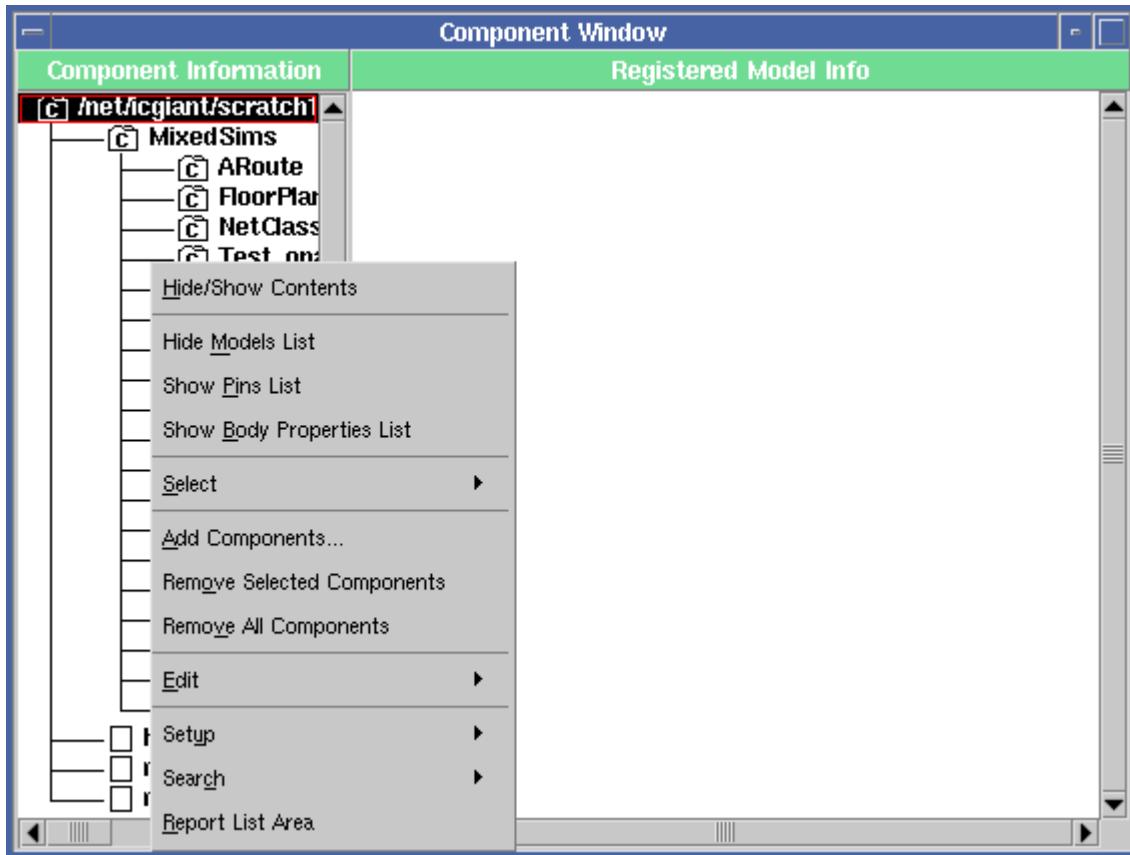
Display Hierarchy Levels

Use the **Show levels** item from the popup menu to control the amount of the design hierarchy you want to view. By default the hierarchy is displayed to the second level. But you can display as much or as little of the hierarchy as you need.

While traversing the hierarchy you can expand and collapse the hierarchy by double clicking on the “current” component using the Left Mouse Button, or you can use the **Hide/Show Hierarchy** from the popup menu. This is a toggle that displays everything below the “current” component only.

The “current” component is designated by the red box that surrounds the component list. Non-primitive components are represented by the diamond symbol before the component name. [3-53](#) illustrates a graphical Design Hierarchy Window with its associated popup.

Figure 3-53. Hierarchy Popup Menu



Set the Current Object

The “current” object is designated by the red box that surrounds one of the objects in the component list. The instance list is associated with this “current” object. The current object can be either the one of the selected objects or none of the selected objects. You can change the “current” object by moving the red box up and down the hierarchy using the arrow keys or selecting with the mouse. When you change the current object the instance list is updated.

Select Objects

You can Select objects using the Left Mouse Button. This adds objects into the object set. To unselect an object hold down the control key while clicking on the Left Mouse Button. To unselect all objects use the “U” stroke.

Open Additional Hierarchy Windows

Use the **Open New Hierarchy** command from the popup to view a separate portion of the design hierarchy or to view another design's hierarchy. A dialog box displays, seeded with the “current” instance name or you can type in the path to an instance. Upon execution of the dialog

box, a new hierarchy window displays. You can perform the same functions in the new window as you could in the old.

Change the Font

The **Setup > Set Font** popup menu item allows you to change the font type, size and boldness of the item names.

Search the Hierarchy

Use **Search** from the popup menu to search forward and backward through the hierarchy. This is essentially the same search mechanism used in the Notepad window. Do **Search > Search** to set the parameters of your search.

Create Reports

Use **Report** from the popup menu to get an indented list of all the information that is currently displayed. You can save this report to an ASCII file or print it directly to any PostScript printer.

Related Topics

[\\$show_component_hierarchy\(\)](#)

Displaying Component Information

You can display component information by using the Component Window. The Component Window allows you to view and/or edit information about specific components in your design.

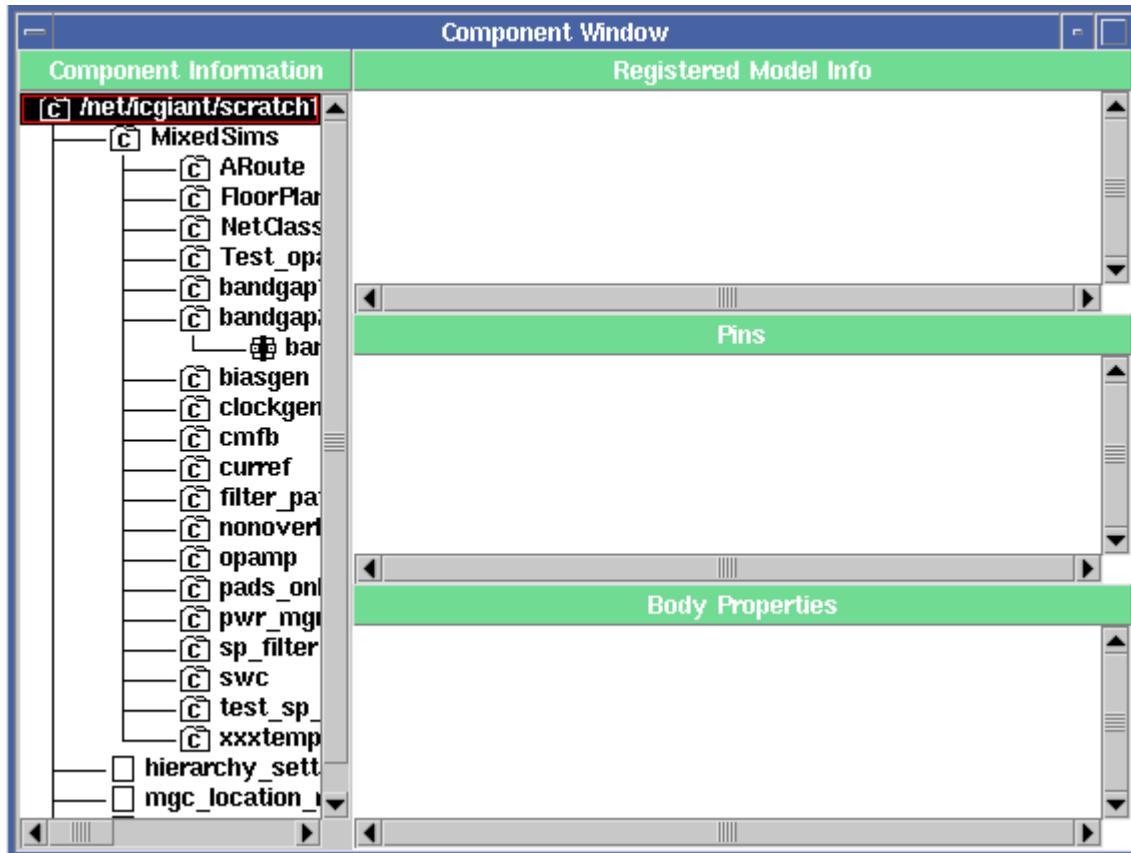
Component Window	294
Component Window Setup	296
Selecting a Component	296
Selection of Part Interfaces	297
Registering a Model	298
Deleting a Part Interface	298
Renaming a Part Interface	298
Setting the Default Part Interface	299
Save, Update or Forget Edits	299
Unregister a Model	299
Validate a Model	300
Adding and Deleting Labels	300
Create Reports on the Models List	300
Displaying the ASCII comment file	301
Printing the Contents of the Component Window List area	301

Component Window

To access: **MGC > Design Management > Open Component Window**

A Window displays divided into four distinct information list areas. Each list area has a separate popup menu.

Figure 3-54. Component Window Display



Fields

- Component Information

Displays an indented list of the component and its contents. Icons next to items indicate the object type. Multiple components can be shown at one time. Filtering of types is available.

- Registered Model Information

Displays all registered models for each part interface selected in the component list area. Labels are shown indented underneath the model name. Models are distinguished for a given component by the gray header bar, and are listed underneath.

The Models List area displays all of the models that are registered for a selected Part Interface. You display this information using the Show Models command from the

popup menu in the Component information window. As you select and unselect part interfaces the model list updates both the models and the labels.

Each Model has a label or multiple labels associated with it. To view the labels for a selected model, either use the Show Labels command or double-click on the model.

By default, the models are grouped by type, as listed in [Table 3-3](#).

Table 3-3. Model Types

Model	Type
Graphic Models	mgc_symbol
Functional Models	mgc_schematic, BLM, QPT, Eddm_single_objec, hdl_arch_do, MTM
Timing Models	Technology, Library Technology, Linear Timing
Other Model Types	Non-MGC standard Model types, hdl_entity_do

The Model list allows you to make several changes without having to access another application. These include the ability to:

- Unregister a Model from the currently selected Part Interface
- Validate a Model
- Add or Delete a Label from the currently selected Model.
- Produce various reports on models
- Pins

Displays Pin names and properties for each selected part interface.

The Pins List area displays all of the Pins for the selected Part Interface. You display this information by using the **Show Pins List** command from the popup menu in the Component information window. By default all pin names are displayed. The Pins List shows the complete name (and the user name, if it is different) of all the pins associated with each pin. Pins are grouped underneath a gray header bar for each Part Interface.

You display the properties for each pin by using the Show Pin Properties command from the popup menu in the Pins List area, or by double-clicking on a pin name.

As with the Model list, the Pins list updates to show pin information for newly selected interfaces.

You can Show/Hide Pin properties using either the Hide Pin Properties or Show Pin Properties toggle command from the popup menu.

- Body Properties

Displays the Body Property name and value for each selected part interface.

As with the Pins and Models list areas, Body Properties are automatically updated when a new Part Interface is selected.

Related Topics

[Component Window](#)

Component Window Setup

You specify the characteristics that control the display of the component window by activating the **Setup** menu item. Options for this item allow you to change the foreground, background and text colors, the size and placement of the windows, and also the size of the font for each text area.

Once you have optimized your setup, save it to a startup file using the **Setup > Write Default Startup File** command from the popup menu. This saves this information in your user's startup file. You can also enter this information into either a site or project file so it can be shared.

Related Topics

[\\$write_default_startup_file\(\)](#)

Selecting a Component

Select a component for viewing.

Procedure

1. Open the Component window using the **MGC > Design Management > Open Component Window** pulldown menu.
2. Select the Component(s) from the Navigator.
3. Specify in the options form the type of list information you want to view.

You can also specify this information from a startup file.

Figure 3-55 shows the component window with only the Model information displayed. This information list shows the component name and the components part interfaces, but it does not show the contained objects. To view these objects, double click on the component to expand its contents. Select the different part interfaces beneath it to cause the Model, Pins and Body Properties list to update immediately.

Figure 3-55. Component Window with No Model Information Showing

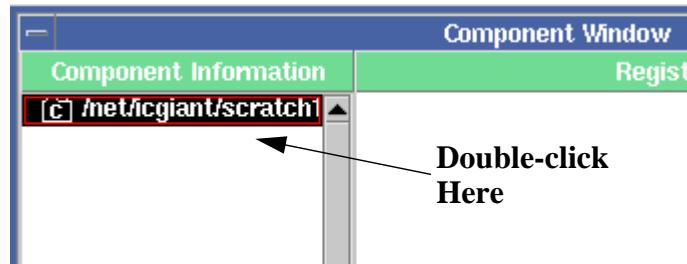
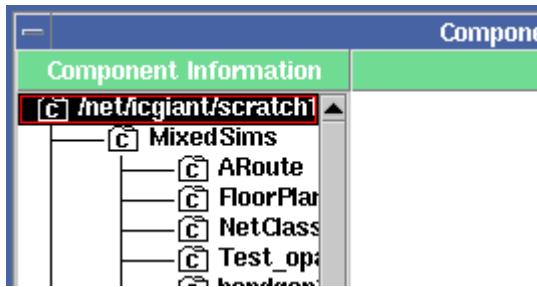


Figure 3-56 illustrates the Component Window with expanded information.

Figure 3-56. Component Window Expanded Information



As previously stated, each list area contains its own popup menu. For example, if you wanted to view the labels associated with the models found in Figure 3-56, you would choose the Show Models List item from the popup menu. The Component Information List

The Component Information List Area, displays every component that is associated with your design. You can use this list area to edit or display the Model, Pin and Body Properties for each of these components. The following discusses the functions that are associated with the Component Information List.

Related Topics

[Component Window Setup](#)

Selection of Part Interfaces

There are a number of ways in which you can select single or multiple Part Interfaces.

If you want to select a single part interface or multiple part interfaces in a single component, use your mouse holding down the select button and fanning over the part interfaces. But if you want to view a particular part interface use the **Select** command from the popup menu.

Related Topics

[Deleting a Part Interface](#)

[Renaming a Part Interface](#)

Registering a Model

By default, all models listed are registered to all part interfaces that are listed. You must first select the part interface and then select the model or models to register with the part interface.

Procedure

1. Select the Part Interfaces and Models you wish to register
2. Choose **Edit > Register Model** from the popup menu
3. Confirm your selections on the resulting form.

Related Topics

[\\$register_models\(\)](#)

Deleting a Part Interface

Delete a selected part interface.

Procedure

1. Use the **Edit > Delete Interfaces** command to delete a selected part interface.
2. Upon deletion the contents of the Models, Pins and Body Properties list are emptied.

Related Topics

[\\$delete_part_interfaces\(\)](#)

Renaming a Part Interface

Rename a part interface.

Procedure

1. Choose **Edit > Rename Interface** from the popup menu.
2. The resulting dialog box allows you to choose whether or not the name change is global to all Components with this Part Interface or if it is only locally applied.

Related Topics

[\\$rename_part_interface\(\)](#)

Setting the Default Part Interface

Change which Part Interface is the default for a given component.

Procedure

1. Use the **Edit > Set Default Part** Interface command from the popup menu.
2. The Default interface is indicated by the label “*(Default)*” following the name of the component. The Default Part Interface can be set for multiple components at one time. That is, when you set the Default Part Interface, any component that contains that Part Interface is listed as the default.

Related Topics

[\\$set_default_part_interface\(\)](#)

Save, Update or Forget Edits

If you are not in a read-only mode, you can either save, update or forget the edits you apply to the Component information list area.

Prior to saving edits, the components you change are indicated by reverse video. Once the save operation is activated the normal video is restored.

You can also forget edits on a particular part interface by selecting that component and issue the Forget/Update Component Edits command from the popup menu. This command forgets the edits and then read in a fresh copy of the component from disk. Updating is useful when a model for the component was changed from another application window.

Related Topics

[\\$save_components_edits\(\)](#)

[\\$forget_components_edits\(\)](#)

Unregister a Model

You can unregister a model from a Part Interface using the Unregister Model command from the popup menu.

Whenever you choose to unregister a model a confirmation box appears. When confirmed, the Model list is automatically updated to show the only those models that are registered.



Caution

Once Models are unregistered you cannot undo the operation. You also lose your pins list if the last occurrence of the model type is unregistered.

Related Topics

[Displaying Component Information](#)

Validate a Model

Validate your models by selecting the model and using the Validate Model command from the popup menu.

The validation process first tries to connect the model by checking for its existence, then performs the validation process. Part of this process is in validating that the model has appropriate pins for the Part Interface. Upon completion a report window displays with the model information.

 **Note** Models of type mgc_symbol cannot be validated from the Component Window.

Related Topics

[\\$validate_models\(\)](#)

Adding and Deleting Labels

You can add a label or multiple labels to a currently selected Model or group of Models.

Procedure

1. Use the **Edit > Add Label** popup menu item.
2. To delete labels, expand the model so that all the labels for that model are showing, and then select the label or labels you wish to delete.

Related Topics

[\\$add_labels_to_models\(\)](#)

[\\$delete_labels_from_models\(\)](#)

Create Reports on the Models List

There are several reports that you can create on the models list.

Each report is displayed as a Notepad window, which you can either print or export to a file. The Model information reports are:

- **Report > List Area** creates a standard report window containing all the models listed for a part interface.

- **Report > Model Validity** gives you a standard report window containing information about the validity of the selected Models. This information is available whether or not you previously performed a Validate Models function.
- **Report > Models by Each Label** creates a standard report window containing a list of all the models which have the selected label. If more than one label is selected in the Models list, then the models are grouped by label in the report window.
- **Report > Models by All Labels** creates a standard report window containing a list of all the Labels for the Model selected.

Related Topics

[\\$report_models_for_each_label\(\)](#)

[\\$report_models_with_all_labels\(\)](#)

Displaying the ASCII comment file

Display a parts comment file.

Procedure

1. From the Component information window you can display a parts comment file (with the component type: Mgc_file) by double-clicking on the comment file. A read-only Notepad window appears.
2. You can then print or export this notepad window if you wish.

Related Topics

[\\$open_read_only_editor\(\)](#)

Printing the Contents of the Component Window List area

Print a report of the Window List area.

Procedure

1. Place the cursor in the list area from which you want a report.
2. Choose the Report command followed by any options that are presented.

A read-only Notepad window appears with the report information. You can copy this information to the clipboard if you wish.

3. From the Notepad window choose **Print > print document**.

You can also export this information to a file using the Export command.

Related Topics

[Displaying Component Information](#)

Design Management in the Operating System Shell

The Pyxis Project Manager provides all of the features that you need to invoke tools and manage your design data.

However, if you choose, you can perform some of these tasks in the operating system shell. If you do invoke your application directly from the shell, you do not have access to the tool viewpoint's pre- and post-processing ability.

 **Tip:** For information on invoking your tool directly from the shell, refer to the “Shell Command Dictionary” in your application's reference manual.

The other major design management task is manipulating design objects. Because design objects are simply groups of files and directories, you can use ordinary operating system commands to copy, move, and delete them. However, **you should not** use ordinary operating system commands to manipulate them. A design object's fileset constantly evolves, and the operating system does not understand which items belong to an object's fileset. As a result, by using operating system commands, you can easily omit a fileset member when attempting to manipulate a design object.

 **Caution**
Do not modify a design object's fileset by using ordinary operating system commands unless you are explicitly instructed to do so in this manual, as part of a salvage procedure. If you use them, you can corrupt your design data to a state that is beyond repair.

The Pyxis Project Manager provides viable, shell-based alternatives to using operating system commands for design management.

Using the Nodisplay Mode	302
Pyxis Project Manager Shell Commands	304

Using the Nodisplay Mode

If you want to use the Pyxis Project Manager's AMPLE-based commands, but do not want the overhead of using the Common User Interface, you can invoke the Pyxis Project Manager in “nodisplay” mode.

With the `-nodisplay` option, the Pyxis Project Manager does not bring up a graphical user interface. Instead, the Pyxis Project Manager remains in the shell, and you supply either a list of AMPLE commands or an input script to it.

Procedure

With the `nodisplay` mode, you have three choices of redirection:

1. Simple `nodisplay` mode

You can invoke the Pyxis Project Manager in `nodisplay` mode from a UNIX shell, as shown in the following example:

```
$ dmgr -nodisplay
```

2. “Here” document

You can execute a ‘here’ document. You might choose to use a ‘here’ document within a script to automate a repetitive Pyxis Project Manager procedure. Here is an example of this type of redirection syntax, invoked from a UNIX shell:

```
$ dmgr -nodisplay <<!
> \$AMPLE_function_1();
> \$AMPLE_function_2();
.
.
.
> !
```

3. Redirection from a script

You can invoke the Pyxis Project Manager in `nodisplay` mode and supply a file as input. Here is an example of this redirection syntax:

```
$ dmgr -nodisplay < your_home/input_file >
```

The Pyxis Project Manager invokes in `nodisplay` mode. After it executes your commands and closes, the results of your commands transcript in the window in which you invoked. Note the following features about invoking in `nodisplay` mode:

- The Pyxis Project Manager startup files execute when the application is invoked in `nodisplay` mode. All transcriptable functions in the startup files transcript in the window in which you invoked.
- You can use function or command syntax for all three redirection methods.
- In the “Here” document example, the backslash (\) that precedes each of the “\$” is required. The backslash prevents the UNIX shell from interpreting the AMPLE functions, whose name typically begins with a dollar sign (\$), as shell variables.
- The functions that you supply to the Pyxis Project Manager in `nodisplay` mode must be available in the scope of the currently active window. When you first invoke in `nodisplay` mode, that window is the Pyxis Project Manager session window. If you open other windows, you can issue commands that are available in only those windows

scopes. However, to issue commands that are available in the other windows, you must first use `$set_active_window()` function to activate the newly opened windows.

Each Pyxis Project Manager function that produces a new window returns the name of the window produced. This is the value that you supply to the `$set_active_window()` function.

- The first time that you change your session setup characteristics in the Pyxis Project Manager session using one of the **Setup** > menu items, the Pyxis Project Manager prompts you about whether to save the changes to your default startup file. If you change your toolbox search path at any time during a session, when you exit the session the Pyxis Project Manager also prompts you whether to save those changes.

When you run a 'here' document by invoking the Pyxis Project Manager with the `-nodosplay` switch, and in the 'here' document you change either the toolbox search path or a session setup characteristic, the Pyxis Project Manager still prompts you whether to save the changes to a startup file.

Because you are running in `nodosplay` mode, neither of these prompts is visible and the Pyxis Project Manager appears to hang while waiting for input. To prevent this from occurring, at the beginning of your Pyxis Project Manager session you should set the values of the AMPLE variables `$MGC_SAVE_SETUP` and `$toolbox_search_path_changed`, as shown in the following example:

```
$ $MGC_SAVE_SETUP=@no
$ $toolbox_search_path_changed=@false
```

Related Topics

[Windows and Scopes](#)

AMPLE for Pyxis User's Manual

[dmgr_ic](#)

Pyxis Project Manager Shell Commands

In addition to an AMPLE interface, the Pyxis Project Manager provides you with shell commands for design management.

These commands execute directly in the operating system shell and allow you to perform basic operations on design objects, such as copy, move, and delete. Unlike UNIX commands, these shell commands understand a design object's type. As a result, these commands always account for all members of a design object's fileset. Table 3-4 summarizes the Pyxis Project Manager shell commands.

Caution



Mentor Graphics highly recommends that you use the Pyxis Project Manager rather than the Pyxis Project Manager shell commands listed in this section. By using the actual Pyxis Project Manager you have more functionality, greater speed, and better ease of use through its graphical interface.

Table 3-4. Pyxis Project Manager Shell Commands

Shell Command	Description
change_references_ic	Changes the reference pathnames and reference version numbers of a design object.
chref_ic	Changes the references in the objects contained within the specified directories.
checkref_ic	Checks the references of all objects found in the specified directories to determine if the version of the object pointed to by each reference exists.
copy_object_ic	Copies a design object to a new location.
copy_version_ic	Copies a single version of a design object to a new location.
ddms_locenv_ic	Reads a location map and defines shell environment variables based on the entries in the map.
ddms_which_map_ic	Goes through the search rules and returns the full pathname of the location map file that would be found and used by an application.
delete_object_ic	Deletes a design object.
dmgr_ic	Invokes the Pyxis Project Manager.
freeze_version_ic	Freezes a version of a design object, preventing it from being deleted due to version depth.
get_hard_name_ic	Converts a list of pathnames to hard pathnames.
get_soft_name_ic	Converts a list of pathnames to soft pathnames.
list_contents_ic	Lists the design objects in a directory or list of directories.
listref_ic	Lists the references of all objects found in the specified directories and their sub-directories.

Table 3-4. Pyxis Project Manager Shell Commands (cont.)

Shell Command	Description
list_references_ic	Lists the references of a design object.
move_object_ic	Moves a design object to a new location.
revert_version_ic	Deletes the current version of a design object and makes the previous version current.
salvage_object_ic	Salvages a damaged design object caused by an application failure.
set_version_depth_ic	Specifies how many versions a design object keeps.
show_object_info_ic	Displays information about a design object.
unfreeze_version_ic	Unfreezes a frozen version of a design object.

Shell Command Limitations

Although convenient to use, the Pyxis Project Manager shell commands have limitations of which you should be aware. The following list describes the features provided by the graphical Pyxis Project Manager, which the shell commands lack:

- You cannot create references, and references are not automatically updated when you copy, move, or rename an object.
- You cannot undo a deletion, as you can with the Pyxis Project Manager trash can.
- Sophisticated release management is not available.
- You cannot edit properties.
- You do not have graphical, intuitive icons to represent design objects.
- You do not have graphical tool invocation, with sophisticated qualification and termination scripts.
- You cannot navigate references.

Related Topics

[Shell Command Dictionary](#)

Customizing Your Design Management Environment

In addition to the design object types in the Mentor Graphics tree, you can create your own types and their icon fonts by using the Pyxis Registrar.

After you create these types, you can register them with the Pyxis Project Manager type manager so that the Pyxis Project Manager can recognize objects of these types.

- i** **Tip:** For instructions about how to create your own types and icons and how to load them into the Pyxis Project Manager, refer to the *Pyxis Registrar User's and Reference Manual* available on SupportNet.

Installing Process Design Kits (PDKs)

Mentor Graphics PDKs are shipped as tar -gzip files and can be easily installed into Pyxis Project Manager.

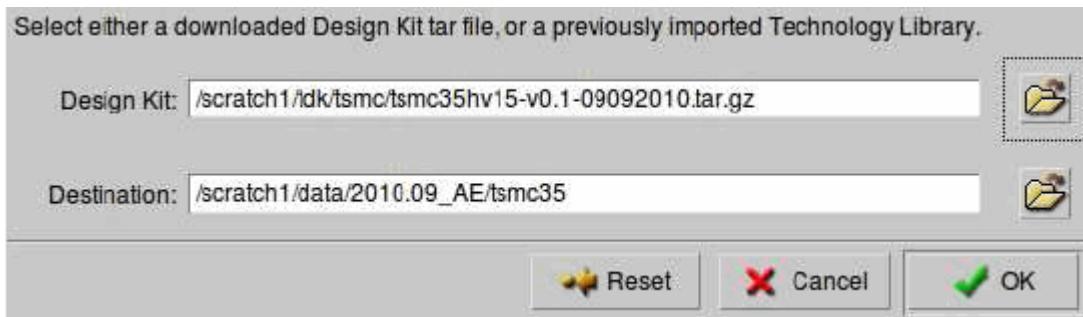
Prerequisites

- Must have a downloaded design kit or previously imported technology library.

Procedure

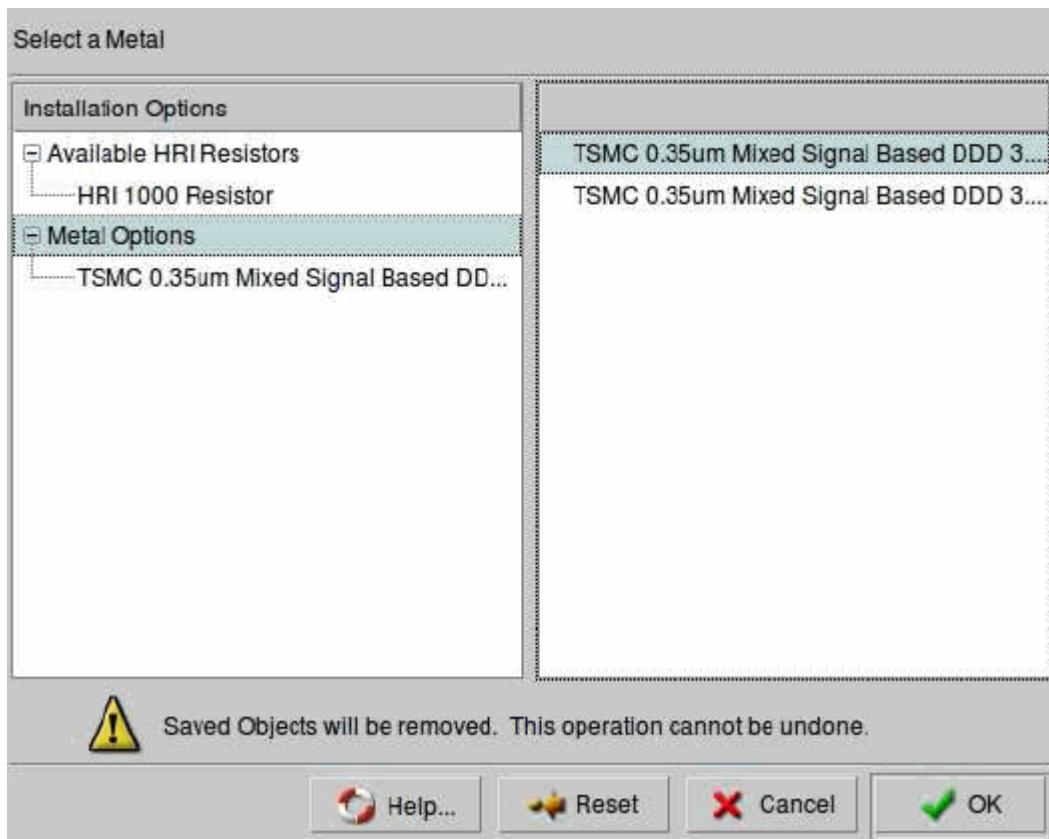
1. Select **File > Install > Process Design Kit**.
2. Browse to the Design Kit location. See the example in [Figure 3-57](#).

Figure 3-57. Process Design Kit Import



3. Choose Installation Options, when appropriate, in the selection form as shown in [Figure 3-58](#). You can choose one set of options per install. You can perform multiple installs if you need multiple options.

Figure 3-58. PDK Installation Options



Related Topics

[Pyxis Process Design Kit User's and Reference Manual](#)

ICanalyst Projects

An ICAnalyst project is associated with the “mgc_ica_project” type. The mgc_ica_project is a container for the ICAnalyst project infrastructure. The instances of the mgc_ica_project data type have an attribute file. The default tool for the mgc_ica_project type is the ICAnalyst application.

Creating a New ICAnalyst Project..... 308

Creating a New ICAnalyst Project

You can create a new ICAnalyst project from the Pyxis Project Manager application.

Prerequisites

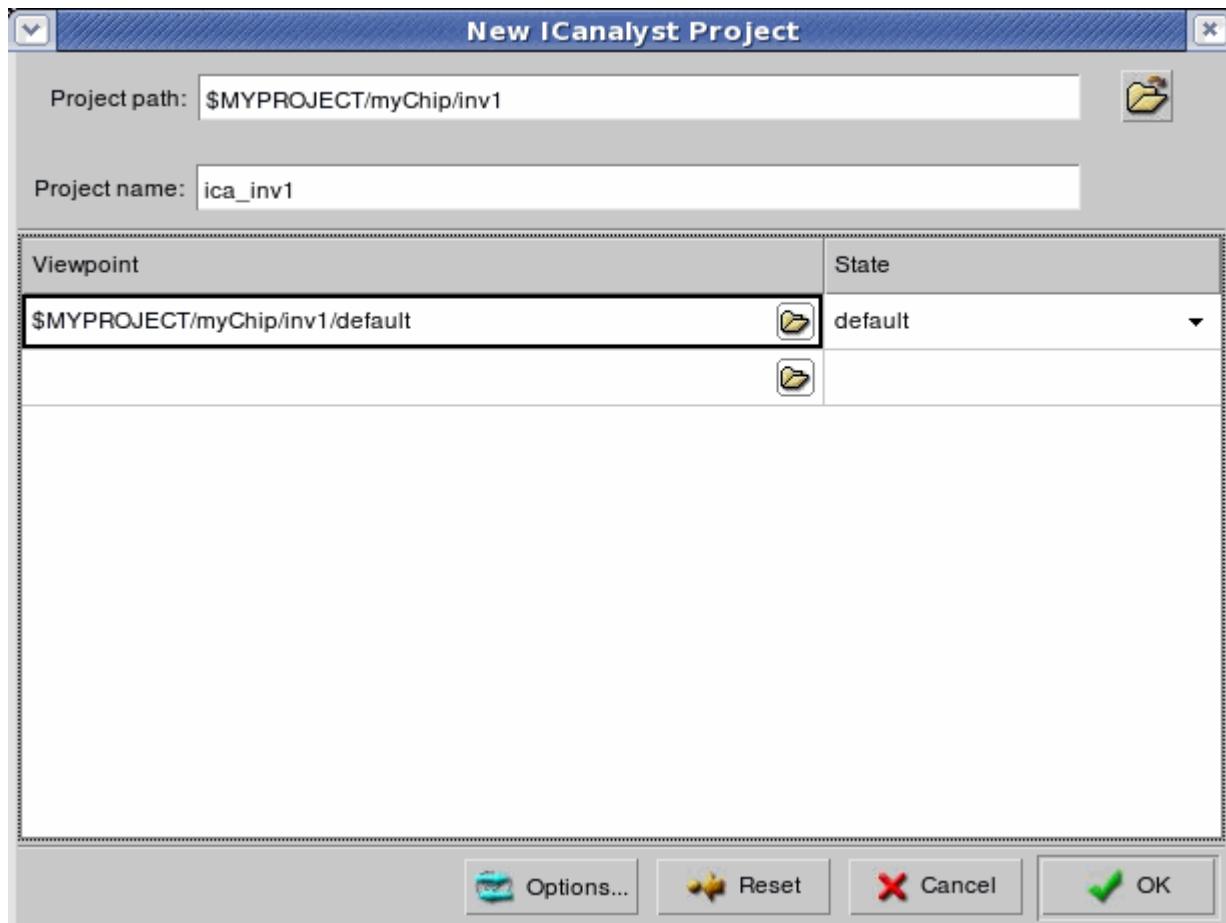
- The Project Navigator window is currently active in the Pyxis Project Manager application.
- The MGC_AMS_HOME value must be set to AMS v15.1 (or later).

Procedure

1. Invoke the New ICAnalyst Project dialog box, as follows:
 - Choose **Tools > ICAnalyst** from the pulldown menu.
- OR
- Select the schematic cell for which you want to create the new ICAnalyst project.
 - From the popup menu for the schematic cell, choose **New > ICAnalyst**.

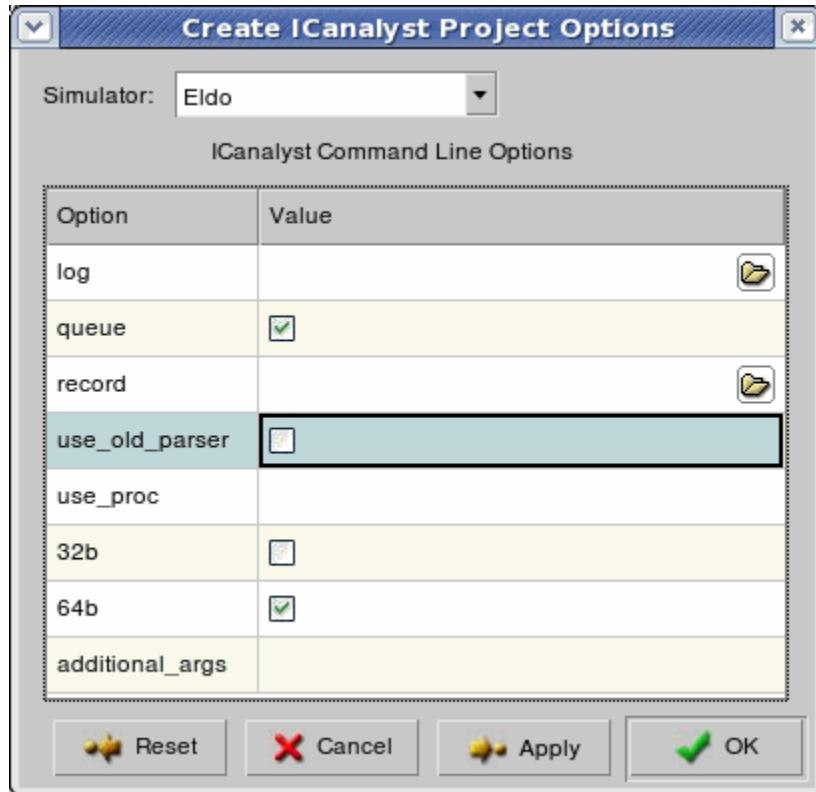
The New ICAnalyst Project dialog box displays, as shown in [Figure 3-59](#).

Figure 3-59. New ICAnalyst Project Dialog Box



2. Specify the project path, project name, and viewpoints for the new ICanalyst project.
3. Click the **Options** button to specify additional ICanalyst command line options for the project in the Create ICanalyst Project Options dialog box, as shown in [Figure 3-60](#).

Figure 3-60. Create ICanalyst Project Options Dialog Box



4. Select the simulator (Eldo, ADMS, or ADiT), and enable or specify the required options.
5. Click **OK** in the Create ICanalyst Project Options dialog box to save the ICanalyst command line project settings.
6. Click **OK** in the New ICanalyst Project dialog box to create the new ICanalyst project with your specifications.

Results

The ICanalyst and Pyxis Schematic applications launch; you can see the newly created ICanalyst project with the settings that you specified.

Related Topics

[ICanalyst Projects](#)

Chapter 4

Project Navigator

Project Navigator is the default window that appears when you invoke Pyxis Project Manager.

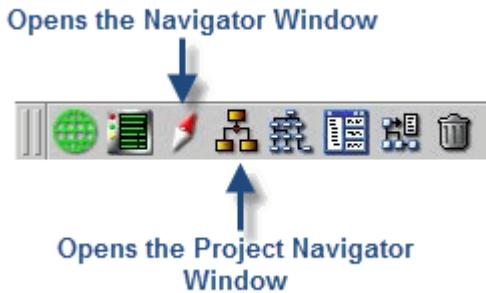
Project Navigator Invocation	311
Project Structure	312
Project Navigator Graphical Interface	316
Project Navigator Hierarchy	320
Project Data and Components	327
Create View Objects	336
Manage Data for a Project	339
External Libraries and Logic Libraries.	348
Manage Technology Settings	352

Project Navigator Invocation

If you have set profiles that do not display Project Navigator as your default window, then the following text discusses the methods of invoking it.

- Pulldown menu — With Pyxis Project Manager launched, navigate from any window to the top menu toolbar. Access the pulldown menu under **Windows > Open Project Navigator to invoke the Project Navigator window**.
- Popup menu — Alternately, from the Project Navigator window, clicking the mouse Menu (right) button opens a context-menu. From that menu, click on the **Open Project Navigator** option.
- Using the toolbar icons — Click on the Project Navigator icon from the Windows toolbar menu, (as shown in [Figure 4-1](#)).

Figure 4-1. Windows Toolbar



Project Structure

Projects are Pyxis Project Manager's solution for organizing all data that is specific to a design project within a single directory hierarchy.

The Pyxis Project Manager project structure consists of a root level project, an MGC location map, one or more libraries, and certain configuration settings.

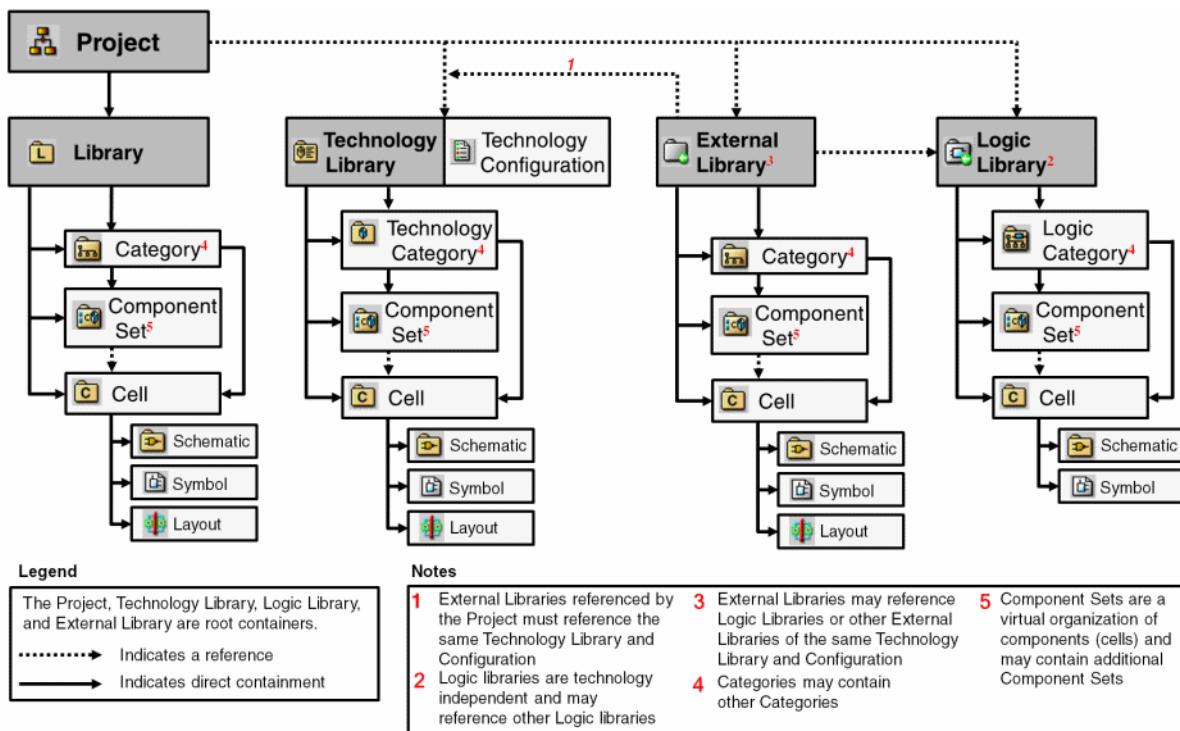
Projects in Pyxis Project Manager are built hierarchically within a well-defined structure. The resident hierarchical components of a project are: **project, technology library, library, category, technology category, and cell**. Some of the project data components, such as the **external library**, reside outside the project.

Note  Before including the external library in a project, Pyxis Project Manager validates the process by comparing the default technology selections for the project and the external library.

Other project data components include the views for the cell design object: **Layout, Symbol, and Schematic**.

[Figure 4-2](#) illustrates the structure for the various design objects associated with the project hierarchy.

Figure 4-2. Structure of the Project Hierarchy



Project	313
External Library	314
Technology Library	315
Library	315
Category	315
Technology Category	316
Cell	316

Project

Projects are Pyxis Project Manager's solution for organizing all data that is specific to a design project within a single directory hierarchy.

The Pyxis Project Manager project structure consists of a root level project, an MGC location map, one or more libraries, and certain configuration settings. The Location Map in turn contains an entry for the project and for each included external library, with the exception of the MGC Standard Libraries, which use a common location map.

Each library may contain any number of categories and cells. Categories may contain additional cells or nested categories. There is no restriction on the depth to which categories may be nested. Cells are the leaf containers in the project structure and may contain any type of view.

A project is a typed directory that can only use a single technology library and technology configuration, the combination of which determines the technology settings for the project.

Technology libraries must be located outside of the project directory structure and are associated by reference.

Projects may also reference, or include, any number of external libraries. All included external libraries must use the same technology settings as the project.

Figure 4-3 illustrates the Project Navigator toolbar. The disabled (grayed out) options are the components that you cannot *directly* create as a child object under the project design object's hierarchy, namely category, technology category, a cell and its views.

Figure 4-3. Project Navigator Toolbar (with project as the active design object)



Tip: For more details on the meaning of the Project Navigator toolbar icons, and what tasks they represent, refer to the section “[Project Navigator Toolbar](#)” on page 316. For more details on creating a new project, refer to the section “[Creating a New Project](#)” on page 331.

Related Topics

[Project Navigator Toolbar](#)

[Creating a New Project](#)

External Library

External libraries are Pyxis Project Manager's solution for organizing standard, re-usable design data within a single data hierarchy.

An external library is a typed directory which contains an MGC Location Map. The Location Map contains a single entry for the external library itself.

An external library may contain categories, cells, its location map, and any configuration settings. In addition, external libraries must reference a single technology library and technology configuration, and may reference any number of included external libraries.

Related Topics

[Creating a New External Library](#)

Technology Library

Technology libraries are Pyxis Project Manager's solution for managing technology-specific data and information such as: the process, rules files, and the most basic symbols and language models.

Technology libraries build on the concepts previously laid forth by design kits by incorporating technology configurations which allow users to create re-usable combinations of technology options.

A technology library is a typed directory which contains an MGC Location Map, and there can only be one per design per project.

Technology libraries must also contain a technology category named “configurations”, which is used to store the technology configurations for that technology library.

Related Topics

[Creating a Technology Library](#)

Library

Libraries provide a high level of categorization for data within a project, and enforce a consistent level of categorization between project contents and project includes.

A library may only be created within a project, and may contain categories and cells.

Related Topics

[Creating a New Library](#)

Category

Categories allow data to be categorized to an arbitrary level in order to manage design complexity.

A category may only be created within a library, external library, or another category. A category may contain nested categories and cells.

Related Topics

[Creating a New Category](#)

Technology Category

A technology category serves the same functional purpose as a regular category, which is to allow categorization of data to an arbitrary level in order to manage design complexity.

However, technology categories have less restrictive containment rules in order to allow greater structural flexibility within technology libraries.

A technology category is a typed directory that may be created within a technology library or another technology category.

A technology category may not directly contain a project, external library, technology library, or library. However, you can create nested technology categories.

Related Topics

[Creating a Technology Category](#)

Cell

Cells organize a related set of design data into a single directory. Each view related to a particular piece of a design is organized within the same cell.

A cell is a typed directory that may be created only within a library, external library, category or technology category.

A cell may not directly contain any other hierarchical containers. However, cells can have any or all of these views: Layout, Symbol and Schematic.

Related Topics

[Creating a New Cell](#)

[Create View Objects](#)

Project Navigator Graphical Interface

The Project Navigator GUI contains toolbars, windows, and popup menus.

Project Navigator Toolbar	316
Project Navigator Window Display	318
Project Navigator Popup Menu	320

Project Navigator Toolbar

The Project Navigator toolbar is available inside an active Project Navigator window.

The icons in the Project Navigator toolbar, and the corresponding actions when you click on any of them in enabled state, are illustrated in [Table 4-1](#):

Table 4-1. Project Navigator Toolbar Icon Description

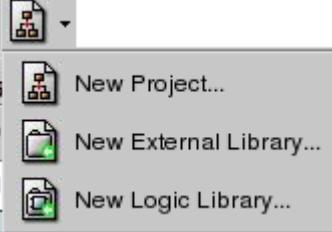
Project Navigator Toolbar Icon	Action upon clicking Icon
	The New Project drop-down icon listing includes three choices which open a corresponding dialog box: New Project, New External Library, and New Logic Library.
	The Open dialog box appears.
	The Create New Library dialog box appears.
	The Create New Category dialog box appears.
	The Create New Cell dialog box appears.
	The Create New Component Set dialog box appears.
	The Create New Layout dialog box appears.
	The Create New Symbol dialog box appears.
	The Create New Schematic dialog box appears.
	The New Language Source dialog box appears.

Table 4-1. Project Navigator Toolbar Icon Description

	Compiles HDL source files. After the compilation is complete, a message is output indicating whether or not the compilation was successful.
	Invokes the Registration Preview dialog enabling you to preview all the models in the selected design unit before running the registration.
	Invokes the Register Model from Source Dialog Box .

Depending on the active hierarchical level of the selected data object, Pyxis Project Manager appropriately enables or disables (grays out) some of the icons in the toolbar set.

Related Topics

[Compiling HDL source file\(s\)](#)

[Registering a Model from a Source File](#)

[Registering All Models from a Source File](#)

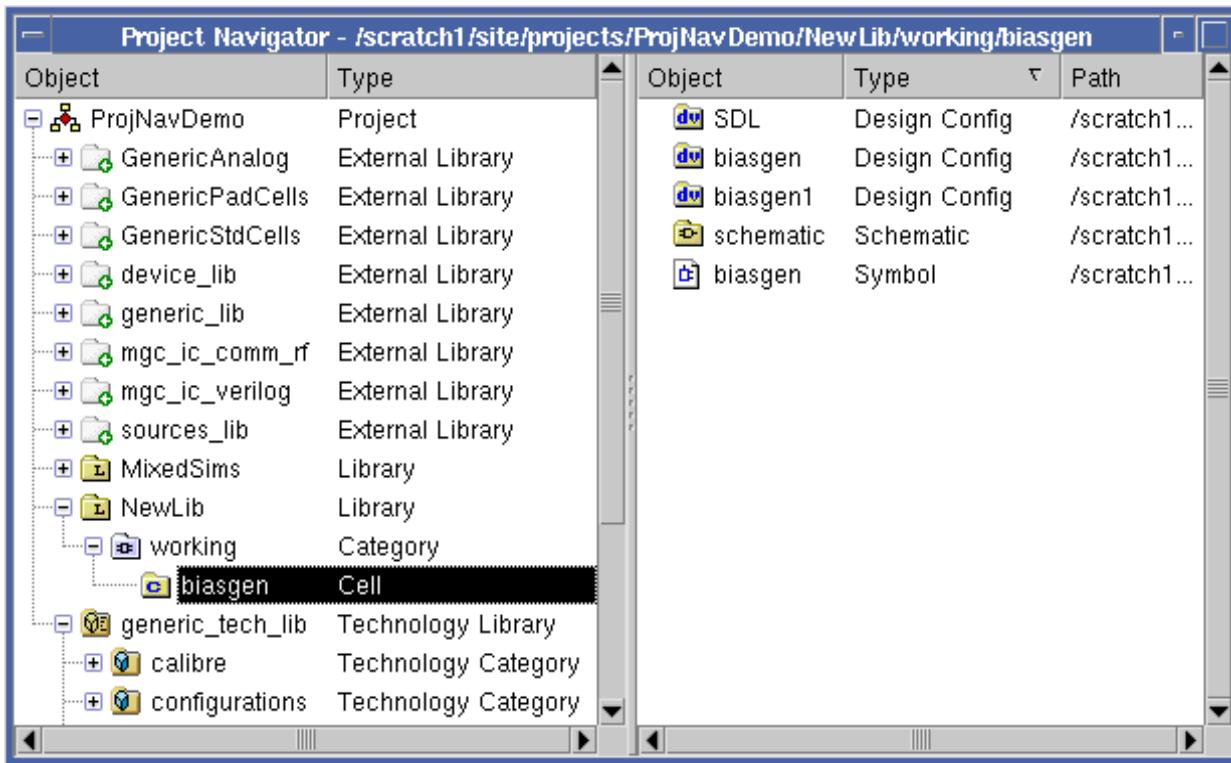
Project Navigator Window Display

The Project Navigator window is split into two portions or panes that function like two display screens with individual scrollbars on their right.

The left pane is the explorer pane, while the right pane is the view pane, as shown in [Figure 4-4](#),

This two-paned explorer/ view window model displays hierarchy on the left in the explorer pane and the contents of the current hierarchical location on the right in the view pane. Both these areas provide a context menu which allows easy access to the common commands for operating on the selected objects.

Figure 4-4. Project Navigator Window



Title Bar

Displays the absolute path for the current hierarchy.

Explorer and View Panes

Opened hierarchies (such as projects, external libraries, and/or technology libraries) are shown in the explorer pane on the left side of the Project Navigator. It should be noted that the Pyxis Project Manager Project Navigator allows multiple hierarchies to be opened simultaneously.

Note

 Only one item may be selected in the explorer pane at a time. The availability of commands may vary based on the type of container that is currently selected.

To view the contents of a particular container in the explorer pane:

- Select the container in the explorer pane and the contents appear in the view pane.
- The list view on the left (explorer) pane only displays the names of all the containers in the Project hierarchy.
- The list view for the right (view) pane displays both the names of the sub-containers as well as the exact directory path at which each of the sub-containers reside.

Here are some mouse navigation options for select tasks within the explorer/view panes:

Task	Mouse navigation
Display/ hide sub-containers in the explorer pane	Click the expander (+ / -) to the left of the container icon, or double-click the container item in the explorer pane.
Select a container or sub-container in either pane	Position the mouse in the pane and quickly type the name of the container that is shown in the pane.

Here are some keyboard navigation options for select tasks within the explorer/view panes:

Task	Keyboard navigation
Display/ hide sub-containers in the explorer pane	Press the '+' (plus) key in the number-keypad.
Expand the entire hierarchy beneath a particular container	Press the '*' (asterisk) key in the number-keypad.
Collapse the entire hierarchy beneath a particular container	Press the '-' (minus) key in the number-keypad.

Related Topics

[Project Navigator Graphical Interface](#)

Project Navigator Popup Menu

The Project Navigator window popup menu can be accessed by either clicking the mouse Menu (right) button, or by clicking the F4 function key.

The popup menu that appears for any hierarchical component is based on the active selection.

Related Topics

[Project Navigator Graphical Interface](#)

Project Navigator Hierarchy

The purpose of the data hierarchy in Project Navigator is to provide users with a structured framework for organizing Project-specific data within a single directory structure.

Project Hierarchy	321
Opening a Hierarchy	321
Copying a Design Hierarchy	323

Viewing the recent history of the hierarchy	326
Closing a Hierarchy	326

Project Hierarchy

The Project hierarchy has the following basic features:

- Organizes data in a structured format:

The data is organized within projects and libraries in a well-defined, hierarchical format. Each component that is created resides in the specified path and within the structure that is defined for it.

- Simplifies data creation and management through the graphical interface:

Convenient pulldown, popup, or toolbar menus can be used to create, view, and manage the data. Opening, deleting, renaming, moving, or copying design objects, as well as adding references to design objects can all be accomplished using the menus.

- Allows multiple hierarchies to be opened:

Data can be copied from one project to another project or an external library. Data can also be moved from one project to another project or an external library.

Note



Project data cannot be shared from within a project hierarchy with any other data hierarchy in Project Navigator.

Related Topics

[Opening a Hierarchy](#)

[Viewing the recent history of the hierarchy](#)

[Copying a Design Hierarchy](#)

[Closing a Hierarchy](#)

Opening a Hierarchy

Once a project has been created, its hierarchy can be kept open or closed.

This is useful when there are multiple existing projects, and you want to view and work with one project at a time in the Project Navigator window.

Procedure

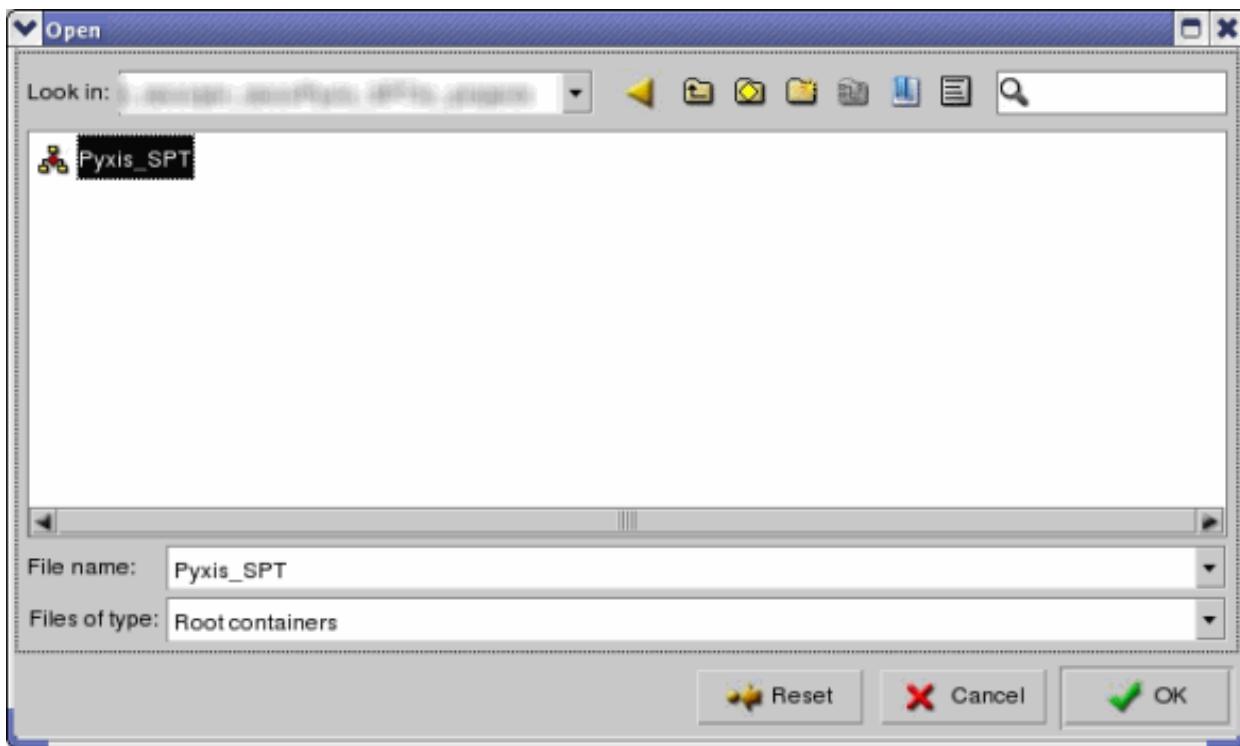
1. Bring up the **Open Hierarchy** dialog box, as shown in [Figure 4-5](#), using any of the following options:
 - Pulldown menu: Access the pulldown menu under **File > Open > Hierarchy**.

- Popup menu: Select the **Open > Hierarchy**, or **Open > Explore Contents**, or the **Open > Explore Contents as Hierarchy** option, depending on the popup menu that appears for the selected hierarchical data object.
- Toolbar icons: Click on the Open icon from the Project Navigator toolbar, as shown in [Table 4-1](#).

Note

Since a cell and its views (i.e. Layout, Symbol, or Schematic) are non-hierarchical data objects, this command only works for all the other (hierarchical) data objects.

Figure 4-5. Open Hierarchy Dialog Box



2. Enter the path for the data object, whose hierarchy you want to view, in the Hierarchy Path field. (Alternately, if you click the Directory Navigator symbol to browse, it displays the Root Hierarchy Navigator window. Click OK to go back to the parent dialog box).
3. Click OK to complete the process.

Related Topics

[Copying a Design Hierarchy](#)

[Closing a Hierarchy](#)

[Viewing the recent history of the hierarchy](#)

Copying a Design Hierarchy

By default, referenced objects coming from external libraries are not copied.

Prerequisites

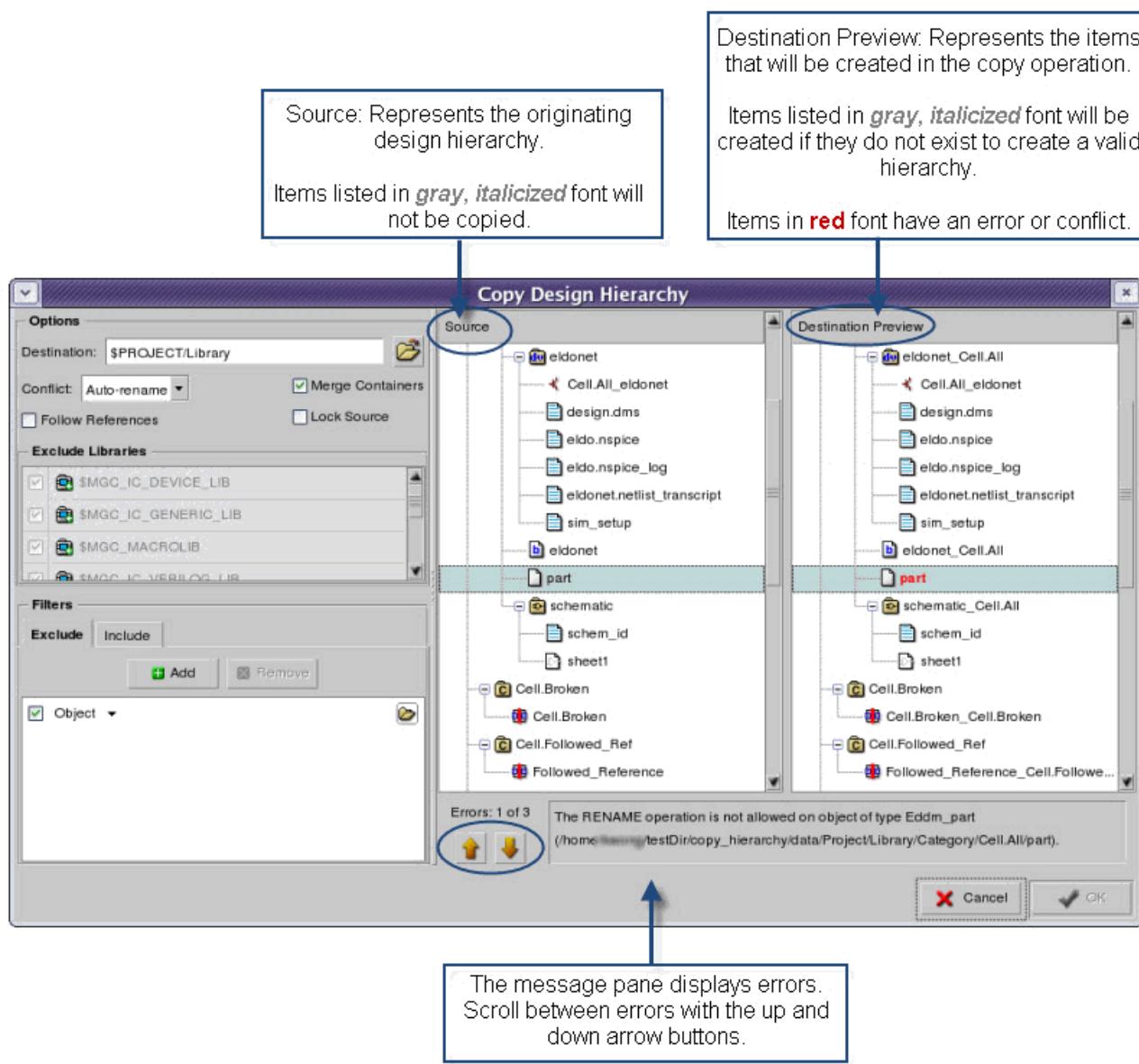
- You have a design hierarchy in Pyxis Project Manager.

Procedure

1. Select **Edit > Copy Design Hierarchy**.

You can also access copy hierarchy functions, as shown in [Figure 4-6](#), using the right mouse button menu. The right mouse menu contains exclude/include objects, exclude/include types, and renaming the destination object.

Figure 4-6. Copy Design Hierarchy Dialog Box

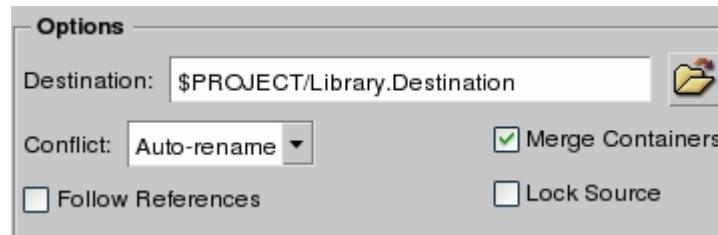


2. Choose **Options** in the upper left-hand side of the dialog box to control the copy destination, conflict policy, and merge containers settings.

You can browse to a copy **Destination** using the file browser button. Available **Conflict** policies include: auto-rename, skip, overwrite, and abort.

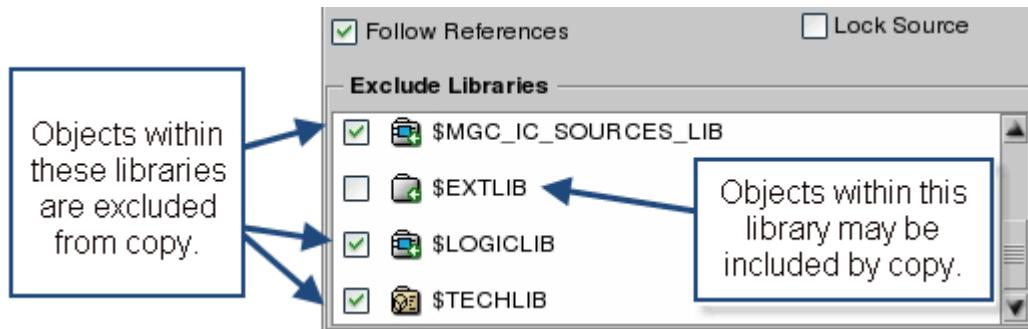
Merge Containers is checked by default. When checked, any basic HDO has its contents merged instead of treating any collisions as conflicts. Refer to [Figure 4-7](#).

Figure 4-7. Copy Design Hierarchy: Options



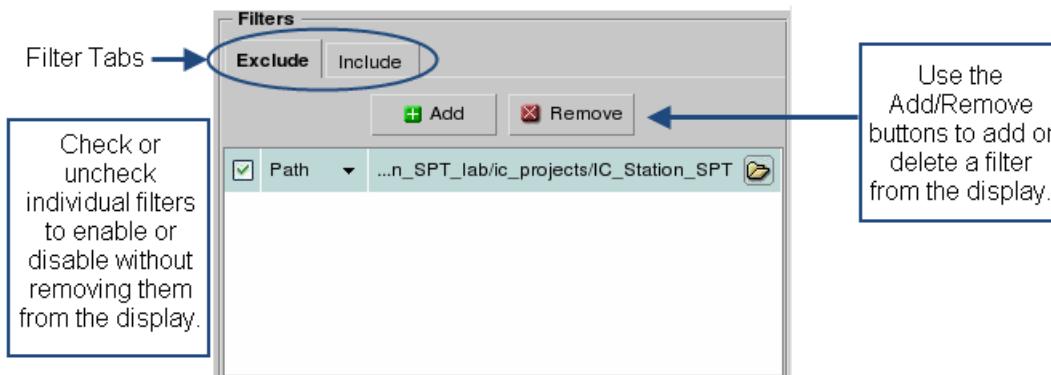
3. Select the **Follow References** checkbox in the **Options** section to copy referenced objects. By default, all objects within external libraries are excluded from copy. Refer to [Figure 4-8](#).

Figure 4-8. Copy Design Hierarchy: Follow References



4. Select Exclude and Include **Filters** in the **Options** section. These filters are displayed in a tabbed format and each filter type contains a list of filters that can be turned on or off. Once you modify a filter, the **Source** and **Destination Preview** panes are updated automatically. Refer to [Figure 4-9](#) and [Figure 4-6](#).

Figure 4-9. Copy Design Hierarchy: Filters



5. Select the **OK** button to perform the copy. If there are errors in the form that cannot be resolved, the **OK** button is disabled.

Results

Your design hierarchy is copied.

Related Topics

[Project Hierarchy](#)

[Opening a Hierarchy](#)

Viewing the recent history of the hierarchy

View the most recent files/ directories that were opened up in the hierarchy.

Procedure

1. Select any object within the active project hierarchy.
2. Access the pulldown menu option under **File > Recent**, which cascades into a list of recently opened projects, technology libraries, and external libraries.
3. Simply clicking on the **File > Recent** option and not selecting any of the options from the cascading menu opens the Open Hierarchy dialog box.
4. Selecting any of the design objects from the cascading menu makes it the current selection within the project hierarchy.

Related Topics

[Opening a Hierarchy](#)

[Closing a Hierarchy](#)

[Copying a Design Hierarchy](#)

Closing a Hierarchy

Close the hierarchy for an open project in Project Navigator.

Procedure

1. Select the hierarchy you want to close.
2. Access the pulldown menu option under **File > Close Hierarchy**.
3. A message is displayed in the Message Area window with a note that the hierarchy has been closed, similar to the one shown below.

Note: Hierarchy
/scratch1/site/projects/ProjNavDemo/NewLib/working/biasgen closed
successfully.

Related Topics

- | | |
|--|---|
| Opening a Hierarchy | Viewing the recent history of the hierarchy |
| Copying a Design Hierarchy | |

Project Data and Components

Once you create a new project, you can add components such as libraries, categories, and cells to the project.

Note

 Before creating a new project, a technology library has to be first created for the project.

Creating a Technology Library	327
Creating a Technology Configuration	328
Creating a Technology Category	331
Creating a New Project	331
Creating a New External Library	332
Creating a New Library	334
Creating a New Category	335
Creating a New Cell	335

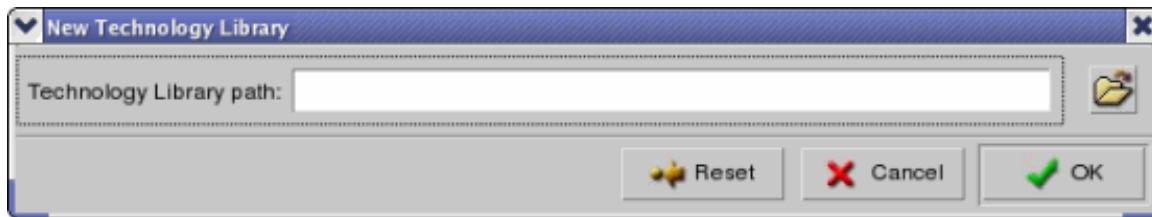
Creating a Technology Library

Create a technology library with the Project Navigator window open.

Procedure

1. Bring up the **Create New Technology Library** dialog box, as shown in [Figure 4-10](#), using the following option:
 - Pulldown menu: Access the pulldown menu under **File > New > Technology Library**.

Figure 4-10. Create New Technology Library Dialog Box



2. Enter the path (where you want all the new technology library to be created), in the Technology Library Path field.

3. Click OK to complete the process.

Related Topics

[Project Data and Components](#)

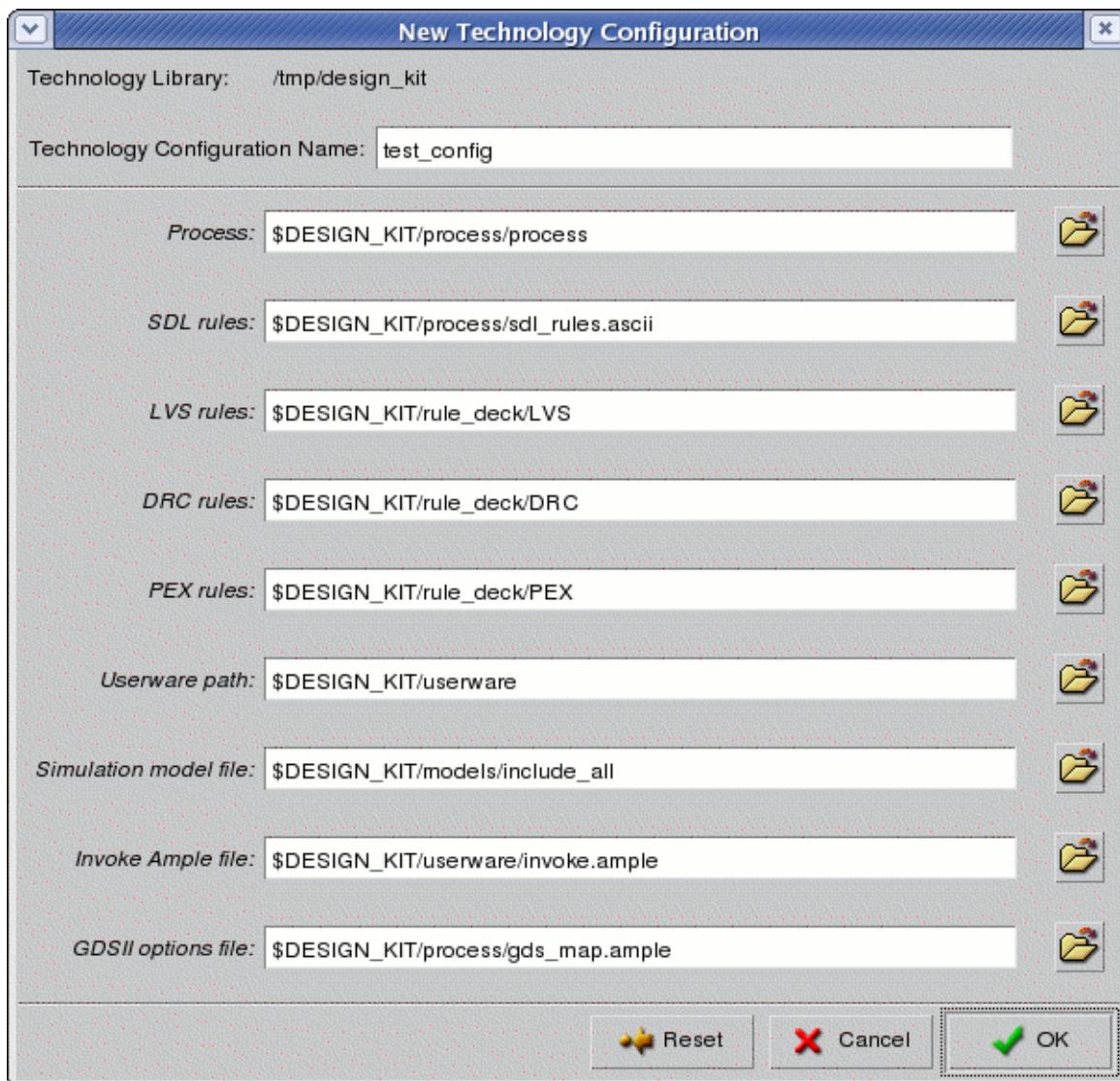
Creating a Technology Configuration

Create a new technology configuration with a technology library selected.

Procedure

1. Select the target technology library.
2. Bring up the **New Technology Configuration** dialog box, as shown in [Figure 4-11](#), using any of the following methods:
 - Pulldown menu: Select the **File > New > Technology Configuration**.
 - Dialogs: Select a Technology library and then click on the  icon in any of the following dialog boxes: **New Project**, **New External Library**, or **Set Technology** dialog boxes.

Figure 4-11. New Technology Configuration Dialog Box



3. Enter the Technology Configuration Name.

Note

There can be multiple technology configurations in the same technology library.

4. Enter the following technology-specific details:

- **Process** (optional)

The process defines technology-specific information, including layers and device generators for Pyxis Layout.

- **SDL Rules** (optional)

The Schematic Driven Layout or SDL Rules file defines how the different layers are connected in Pyxis Layout.

- **LVS Rules** (optional)

The Layout Versus Schematic or LVS rule deck is used by Calibre LVS.

- **DRC Rules** (optional)

The Design Rule Check or DRC rule deck is used by Calibre DRC.

- **PEX Rules** (optional)

The Parasitic Extraction or PEX rule deck is used by Calibre xRC.

- **Userware path** (optional)

The userware path defines the base userware to be used by the tools that are launched from the project.

- **Simulation model file** (optional)

Specifies the SPICE file that you want to include in the simulation. The simulation file defines simulation model parameters and SPICE parameters that affect the behavior of basic SPICE models used in the design. This path is used in Pyxis Schematic's simulation settings for new design viewpoints.

- **Invoke Ample file** (optional)

Specifies the location of the AMPLE file to run when a tool is invoked. Refer to [“Invoking with the MGC_INVOKE_SETUP Variable”](#) on page 163 for more details on setting up the environment for the invoked tool.

- **GDSII options file** (optional)

Specifies the GDSII options file that is used to setup the parameters for Pyxis Layout's Read GDSII and Write GDSII dialog boxes.

5. Click the OK button to create the new technology configuration.



Note

To edit an existing technology configuration, navigate to the “configurations” technology category, and double-click on the technology configuration. This brings up the **Edit Technology Configuration** dialog box with the existing fields pre-populated. Make your modifications and click OK to execute the dialog box.

Related Topics

[Project Data and Components](#)

Creating a Technology Category

Create a new technology category with the Project Navigator window open.

Procedure

1. Select the target technology library or technology category.

Note

To create a technology category, the cursor has to be positioned at the technology library or technology category level of the hierarchy.

2. Bring up the **Create New Technology Category** dialog box, as shown in [Figure 4-12](#), using any of the following methods:
 - Pulldown menu: Access the pulldown menu under **File > New > Technology Category**.
 - Popup menu: Select the **New > Technology Category** option.

Figure 4-12. Create New Technology Category Dialog Box



3. Enter the object name for which you want the technology category to be created.
4. Click OK to complete the process.

Related Topics

[Project Data and Components](#)

Creating a New Project

Create a new project with the Project Navigator window open.

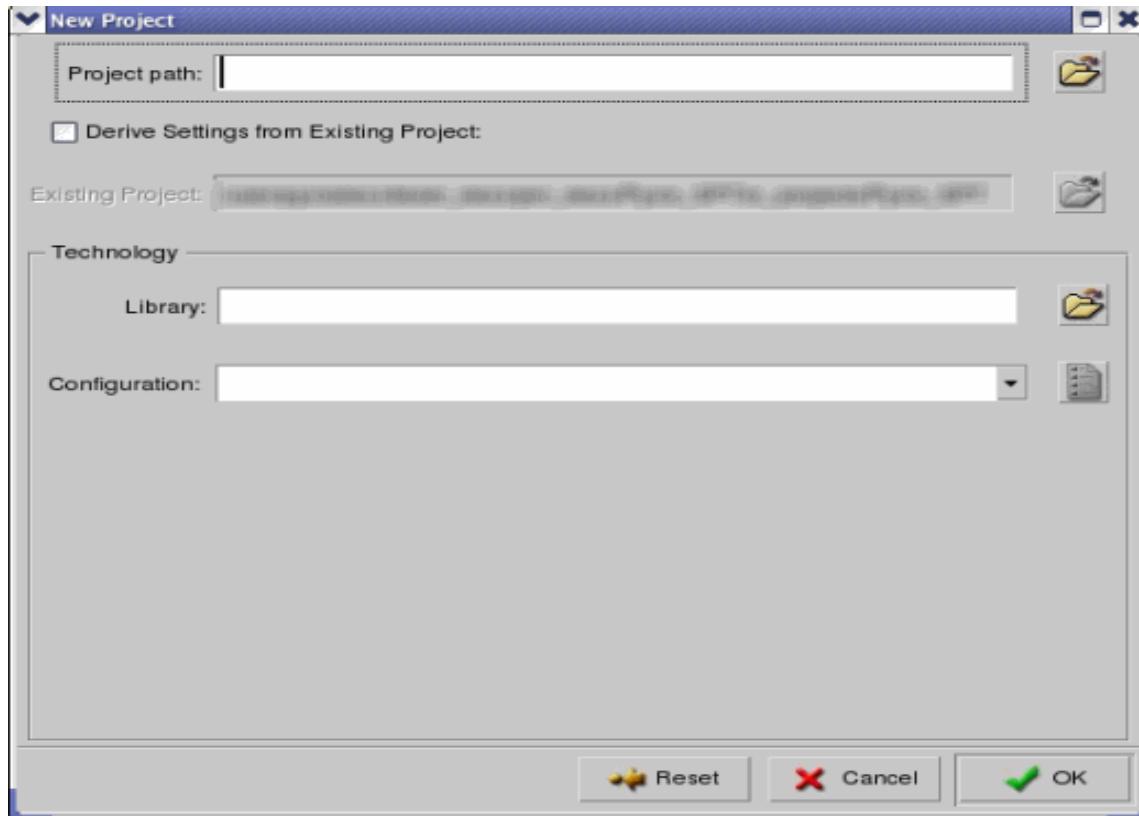
Prerequisites

- Confirm that a new technology library exists before you create a new project.

Procedure

1. Bring up the **New Project** dialog box, as shown in [Figure 4-13](#), using any of the following methods:
 - Pulldown menu: Access the pulldown menu under **File > New > Project**.
 - Toolbar icons: Click on the Project icon (as shown in [Table 4-1](#)), from the Project Navigator toolbar.

Figure 4-13. New Project Dialog Box



2. Enter the path (where you want all the technologies for the project to reside) in the Technology Library Path field.
3. Click OK to complete the process.

Related Topics

[Project Data and Components](#)

Creating a New External Library

Create an external library with the Project Navigator window open.

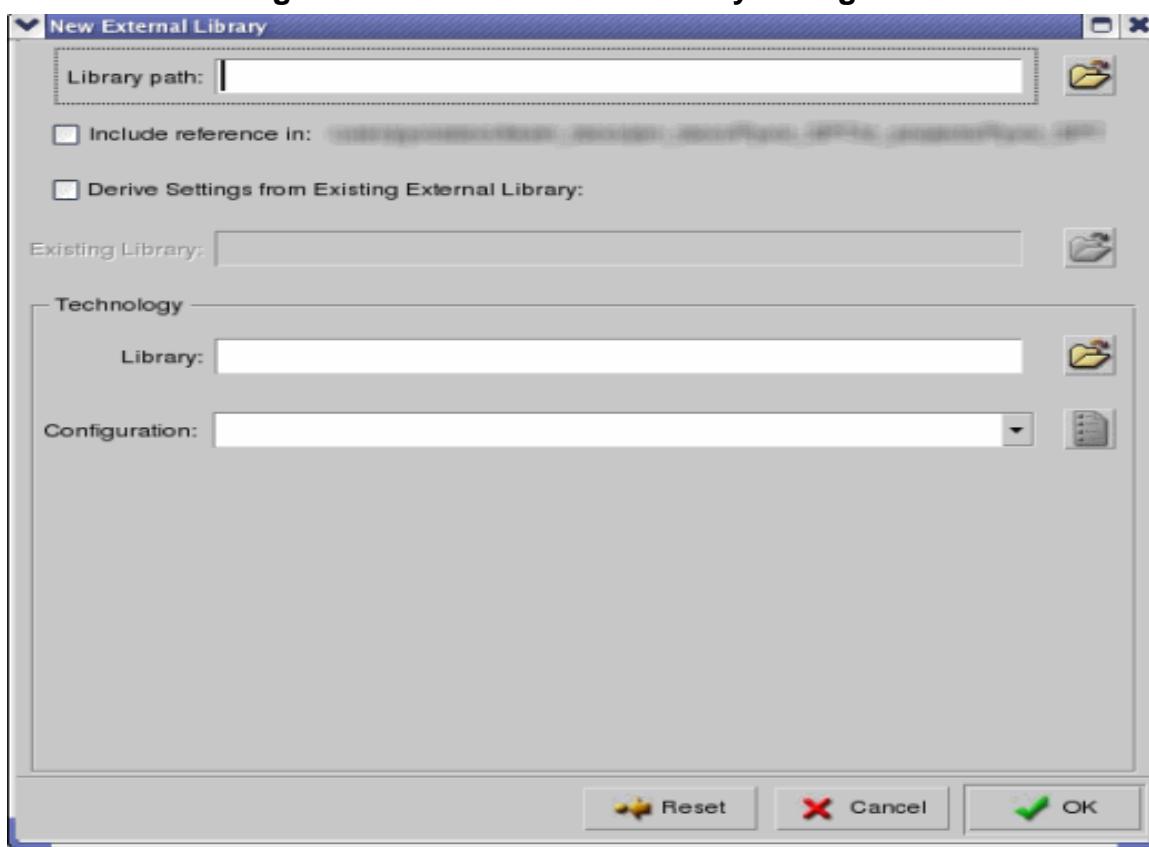
Prerequisites

- Confirm that a new technology library exists before you create a new project.

Procedure

1. Bring up the **New External Library** dialog box, as shown in [Figure 4-14](#), using any of the following methods:
 - Pulldown menu: Access the pulldown menu under **File > New > External Library**.
 - Toolbar icons: Click on the New external Library icon (as shown in [Table 4-1](#)), from the Project Navigator toolbar.

Figure 4-14. New External Library Dialog Box



2. Enter the path (where you want the external library for the project to reside) in the external library Path field.
3. Enter the path for the corresponding technology library in the technology library field. (Alternately, if you click the Directory Navigator symbol to browse, it displays the Technology Library Navigator window.)
4. Select the technology configuration from the drop-down menu in the Technology Configuration field.

5. Check the *Include Standard MGC Libraries* option, if you require the MGC Standard Libraries to be part of this project.
6. Click Ok to complete the process.

Related Topics

[Project Data and Components](#)

Creating a New Library

Create a new library with the Project Navigator window open.

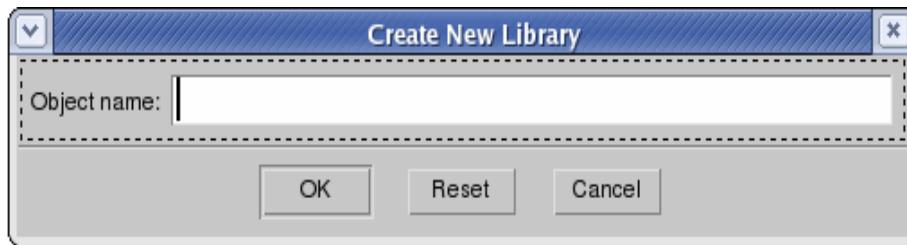
Procedure

1. Select the destination project.

 **Note** To create a library, the cursor has to be positioned at the project level of the hierarchy,

2. Bring up the **Create New Library** dialog box, as shown in [Figure 4-15](#), using any of the following methods:
 - Pulldown menu: Access the pulldown menu under **File > New > Library**.
 - Popup menu: Select the **New > Library** option.
 - Toolbar icons: Click on the Library icon (as shown in [Table 4-1](#)), from the Project Navigator toolbar.

Figure 4-15. Create New Library Dialog Box



3. Enter the object name for which you want the library to be created.
4. Click OK to complete the process.

Related Topics

[Project Data and Components](#)

Creating a New Category

Create a new category with the Project Navigator window open.

Procedure

1. Select the target library, external library, or category.



Note

To create a category, the cursor has to be positioned at the library, external library, or category levels of the hierarchy,

2. Bring up the **Create New Category** dialog box using any of the following methods:
 - Pulldown menu: Access the pulldown menu under **File > New > Category**.
 - Popup menu: Select the **New > Category** option.
 - Toolbar icons: Click on the category icon (as shown in [Table 4-1](#)), from the Project Navigator toolbar.
3. Enter the object name for which you want the category to be created.
4. Click **OK** to complete the process.

Related Topics

[Project Data and Components](#)

Creating a New Cell

Create a new cell with the Project Navigator window open.

Procedure

1. Select the target library, external library, category, or technology category.



Note

To create a cell, the cursor has to be positioned at the library, external library, category, or technology category levels of the hierarchy,

2. Bring up the **Create New Cell** dialog box using any of the following methods:
 - Pulldown menu: Access the pulldown menu under **File > New > Cell**.
 - Popup menu: Select the **New > Cell** option.

- Toolbar icons: Click on the cell icon (as shown in [Table 4-1](#)), from the Project Navigator toolbar.
3. Enter the object name for which you want the cell to be created.
 4. Click OK to complete the process.

Related Topics

[Project Data and Components](#)

Create View Objects

Pyxis Project Manager directly supports creating the following views: Layout (physical view), Symbol (logical view), and Schematic (logical view).

Creating a New Layout	336
Creating a New Symbol	337
Creating a New Schematic	338
Specifying a New Language Source	338

Creating a New Layout

Create a new Layout with the Project Navigator window open.

Procedure

1. Select the target library, external library, technology category, category, or cell.
2. Bring up the **Create New Layout** dialog box using any of the following methods:
 - Pulldown menu: Access the pulldown menu under **File > New > Layout**.
 - Popup menu: Select the **New > Layout** option.
 - Toolbar icons: Click on the Layout icon (as shown in [Table 4-1](#)), from the Project Navigator toolbar.
3. In the **Layout name** field, enter the view name for which you want the Layout to be created.
4. In the **Cell Name** field, enter the cell name for which you want the Layout to be created. If the cell is the active selection, then this field gets pre-populated, and only needs to be verified.
5. Click OK to complete the process.
6. The Layout is opened up in Pyxis Layout.

Related Topics

[Create View Objects](#)

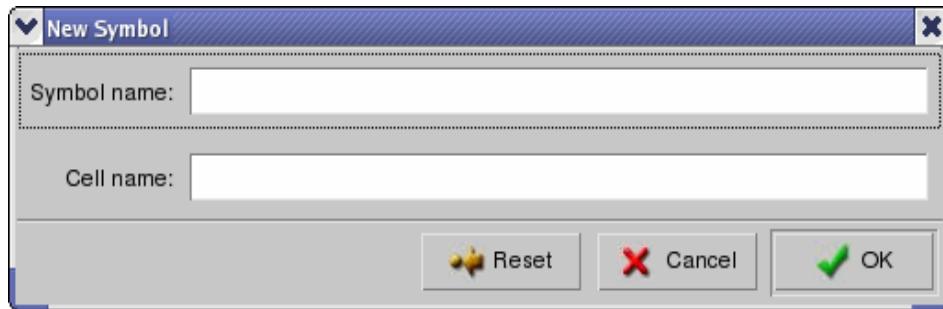
Creating a New Symbol

Create a new Symbol with the Project Navigator window open.

Procedure

1. Select the target library, external library, technology category, category, or cell.
2. Bring up the **Create New Symbol** dialog box, as shown in [Figure 4-16](#), using any of the following methods:
 - Pulldown menu: Access the pulldown menu under **File > New > Symbol**.
 - Popup menu: Select the **New > Symbol** option.
 - Toolbar icons: Click on the Symbol icon (as shown in [Table 4-1](#)), from the Project Navigator toolbar.

Figure 4-16. Create New Symbol Dialog Box



3. In the **Symbol name** field, enter the view name for which you want the Symbol to be created.
4. In the **Cell Name** field, enter the cell name for which you want the Symbol to be created. If the cell is the active selection, then this field gets pre-populated, and only needs to be verified.
5. Click OK to complete the process.
6. The Symbol is opened up in Pyxis Schematic.

Related Topics

[Create View Objects](#)

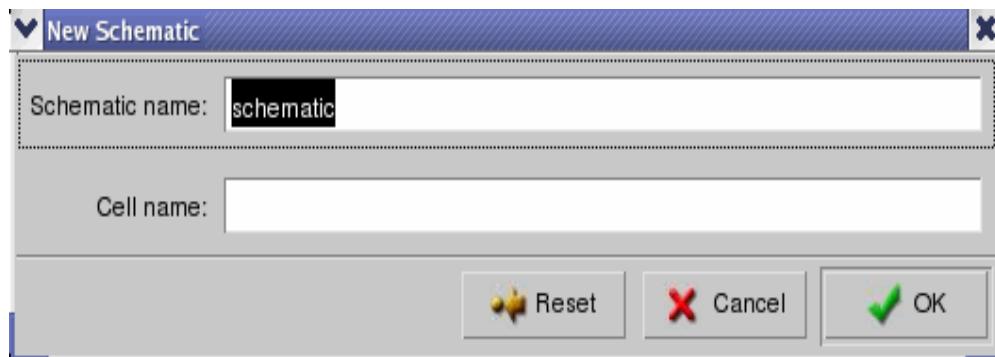
Creating a New Schematic

Create a new Schematic with the Project Navigator window open.

Procedure

1. Select the target library, external library, technology category, category, or cell.
2. Bring up the **New Schematic** dialog box, as shown in [Figure 4-17](#), using any of the following methods:
 - Pulldown menu: Access the pulldown menu under **File > New > Schematic**.
 - Popup menu: Select the **New > Schematic** option.
 - Toolbar icons: Click on the Schematic icon (as shown in [Table 4-1](#)), from the Project Navigator toolbar.

Figure 4-17. New Schematic Dialog Box



3. In the **Schematic name** field, enter the view name for which you want the Schematic to be created.
4. In the **Cell Name** field, enter the cell name for which you want the Schematic to be created. If the cell is the active selection, then this field gets pre-populated, and only needs to be verified.
5. Click OK to complete the process.
6. The Schematic is opened up in Pyxis Schematic.

Related Topics

[Create View Objects](#)

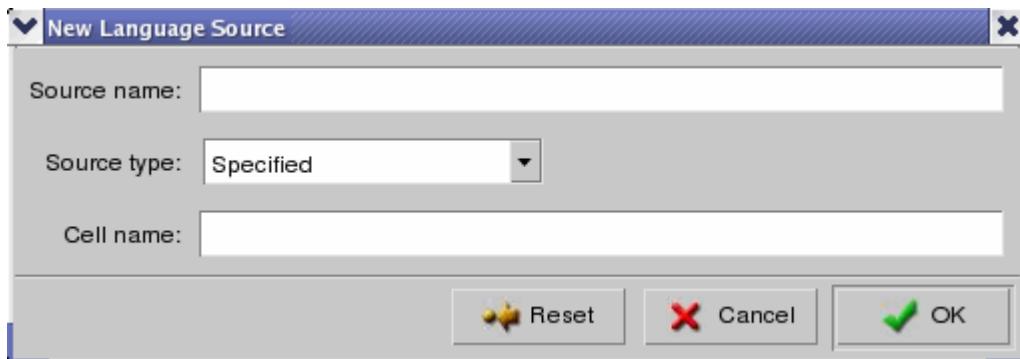
Specifying a New Language Source

Specify a new language source with the Project Navigator window open.

Procedure

1. Select the target library, external library, technology category, category, or cell.
2. Bring up the New Language Source dialog box by selecting the toolbar icon as shown in [Table 4-1](#) from the Project Navigator toolbar.

Figure 4-18. New Language Source Dialog Box



3. In the **Source name** field, specify a name for the new language source. This field may be pre-populated.
4. In the **Source type** drop-down field, select the appropriate type from the list. Available choices include: Verilog (v), Verilog A (va), Verilog AMS (vams), System Verilog (sv), VHDL (vhd), VHDL AMS (vhda), and Spice (spi).
5. In the **Cell name** field, enter a cell name for which you want the new language source to be created. This field may be pre-populated.
6. Click OK to create the process.

Related Topics

[Create View Objects](#)

Manage Data for a Project

Existing projects require you to manage the data components and perform data management tasks copying, moving, deleting, and renaming design objects.

Copying Design Objects	340
Moving Design Objects	343
Renaming Design Objects	346
Deleting Design Objects	347

Copying Design Objects

The copy operation duplicates one or more selected design objects, leaving the selected design objects intact and placing duplicates in the specified destination directory.

When you copy design objects that refer to other design objects which are being copied in the same operation, references between them are updated to reflect their new location.

The copy operation is performed by executing the **Copy Object** command, which can be accessed through the pulldown or popup menu, or by typing the command in the popup command line.

On execution of the **Copy Object** command, the specified object is copied to the destination. The precise outcome may vary depending on the conditions described below:

- If necessary, Pyxis Project Manager opens the copied object in order to select it.
- If the object is an hierarchical design object, all the design objects that it contains are also copied to the destination.
- If the design object selected at the source is a View object and the destination is not a cell, an attempt is made to replicate the parent cell at the destination. Its peer views are not copied unless they are required to maintain the validity of the specified View.

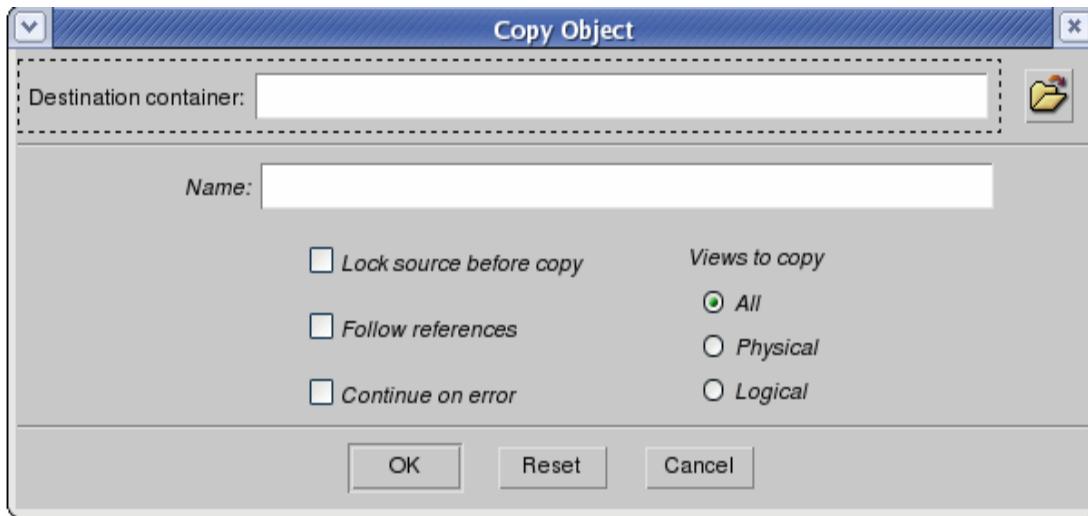
However, if the design object selected at the source is a View object, and you have specified to filter out views of that type, then the View is not copied. The view's cell is not copied unless other views within that cell are also copied

- Any conflicting hierarchical design objects are merged, and any conflicting views are reported as conflicts.
- Pyxis Project Manager reports a summary of the success or failure of the copy operation.

Procedure

1. Select the object that you want copied to a different location.
2. Bring up the **Copy Object** dialog box, as shown in [Figure 4-19](#), using a menu or the command prompt, in any of the following ways:
 - Command prompt: Type the **Copy Object** command in the command prompt bar.
 - Pulldown menu: Access the pulldown menu under **Edit > Copy To**.
 - Popup menu: Select the **Edit > Copy To** option.

Figure 4-19. Copy Object Dialog Box



3. The options have the following meaning:

a. **Destination Container**

Type in the destination of the design object or browse to it.

b. **Name** (optional)

Type in the new name for the destination file in the Name field. If not specified, the source file name is used for the destination file name.

c. **Lock source before copy**

By checking this option, the source design objects are locked before the copy operation is executed, and automatically unlocked at completion.

d. **Follow references**

With this option, you specify whether reference traversal is turned on or off.

Reference traversal determines whether externally referenced design objects of the selected design objects are included in the copy. By default, containment traversal is always set to on, which means that a design object's contained objects are included in the copy.

If Follow References is specified, and if any View being copied references a View that is not in the set being copied (but within the same root hierarchy as the source), then the referenced views and the hierarchical design objects above them are included in the copied operation. Any conflicting hierarchical objects are merged and conflicting views are reported as conflicts.

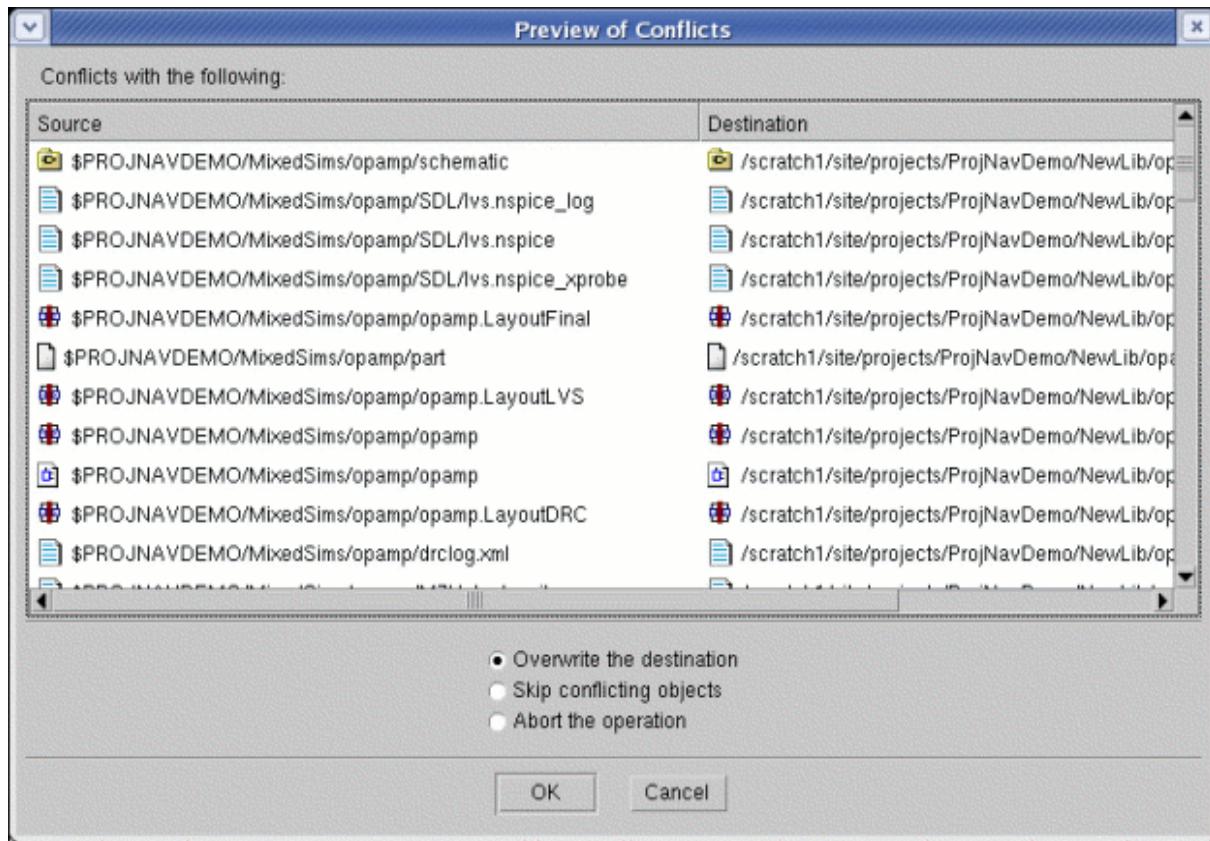
e. **Continue on error**

If any hierarchical design objects being copied at the source conflict with existing hierarchical design objects at the destination, then the contents of both their directories are merged. This occurs recursively until it reaches the View level.

But if conflicting views are found, and the Continue on error option is checked, then the copy process continues uninterrupted, and Pyxis Project Manager reports a complete listing of any objects not copied.

Therefore, in order to resolve conflicts during the copy operation, do *not* check the **Continue on error** option so that when a copy conflict arises, the Preview of Conflicts dialog box can appear, as shown in [Figure 4-20](#), to assist you with resolving the copy conflicts.

Figure 4-20. Preview of Conflicts Dialog Box (for Copy Operation)



The Preview of Conflicts dialog box displays a list of the conflicts and the following choices:

- Overwrite the destination: The conflicting destination view objects are overwritten by the corresponding view objects at the source, and the copy operation continues.
- Skip conflicting objects: The copy of conflicting source view objects to the destination are skipped, and the copy operation continues.

- Abort the operation: The copy operation is aborted.

Your selected choice is applied to all conflicting views. Click OK to execute the Preview of Conflicts dialog box.

f. Views to copy

With this option, select one of the available choices: All, Physical, and Logical. The All option copies all options including physical views, logical views, language and viewpoints. The Physical option only copies the Layout views for the design objects. The Logical option copies the schematic and symbol views for the design object.

4. Click OK to execute the Copy Object dialog box. The source object is copied in to the specified destination directory.

Related Topics

[Manage Data for a Project](#)

Moving Design Objects

The move operation relocates one or more design objects to the specified destination directory.

When there are references between objects being moved simultaneously, those references are updated to reflect their new location.

Automatic reference updating occurs only when you move the related design objects in the same operation. For example, if design objects A and B reference each other, to move them to a new location and update their references to reflect that location, you must move them both in the same operation. Whenever an object is moved from the source to the destination directory, any broken references are fixed within the hierarchy.

The move operation is performed by executing the Move Object command, which can be accessed through the pulldown or popup menu, or by typing the command in the popup command line.

On execution of the Move Object command, the specified object is moved to the destination. The precise outcome may vary depending on the conditions described below:

The specified object, and contents if its an HDO, is moved to the destination.

- If the object is an hierarchical design object, all the design objects that it contains are moved to the destination.
- If the object is a root hierarchical design object, (such as a project, technology library, or external library), its location map is also updated to reflect its new location.
- If the object is a View object, the destination must be an existing cell. Its peer views are not moved unless they are required to maintain the validity of the specified View.

- If any hierarchical design objects being moved at the source conflict with existing hierarchical design objects at the destination, then the contents of their directories are merged. This occurs recursively until the View level.
- If any hierarchical design objects being moved at the source conflicts with an existing hierarchical design object of a different type at the destination, then it is considered an error and the operation fails.
- If conflicting views are found, they are reported as conflicts.
- Pyxis Project Manager reports a summary of the success or failure of the move operation.

Procedure

1. Select the object that you want moved to a different location.
2. Bring up the **Move Object** dialog box, as shown in [Figure 4-21](#), using a menu or the command prompt, in any of the following ways:
 - Command prompt: Type the Move Object command in the command prompt bar.
 - Pulldown menu: Access the pulldown menu under **Edit > Move To**.
 - Popup menu: Select the **Edit > Move To** option.

Figure 4-21. Move Object Dialog Box



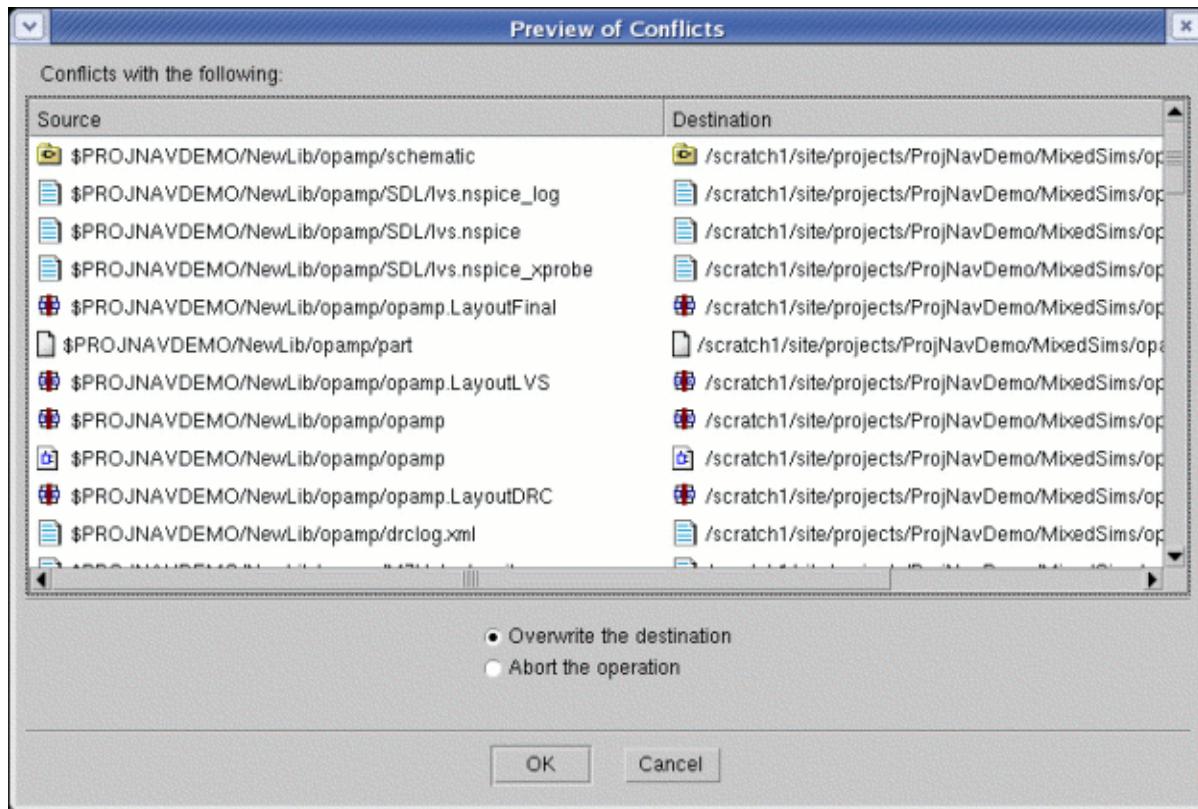
3. The options have the following meaning:
 - a. **Destination Container**
Type in the destination of the design object or browse to it.
 - b. **Name** (optional)
Type in the new name for the destination file in the Name field. If not specified, the source file name is used for the destination file name.
 - c. **Lock source before move**

With this option checked, the target design object is locked before attempting to move it to its new location. This option is enabled by default.

Although not checking this option allows the move operation to execute slightly faster, you should only use this option when you are absolutely sure that the target design objects are not changed by another user during its execution. If the source design object is modified while this operation is executing, your data could be corrupted.

4. If there are no conflicts, or when any conflicts are resolved, click OK to execute the Move Object dialog box. The source object is successfully moved to the specified destination directory.
5. When there are conflicting views during the move operation, the Preview of Conflicts dialog box appears, as shown in [Figure 4-22](#), to assist you with resolving the move conflicts.

Figure 4-22. Preview of Conflicts Dialog Box (for Move Operation)



The Preview of Conflicts dialog box displays a list of the conflicts and the following choices:

- Overwrite the destination: The conflicting destination view objects are overwritten by the corresponding view objects at the source, and the move operation continues.

- Abort the operation: The move operation is aborted.

Your selected choice is applied to all conflicting views. Click OK to execute the Preview of Conflicts dialog box.

Related Topics

[Manage Data for a Project](#)

Renaming Design Objects

Renaming changes the name of a single design object.

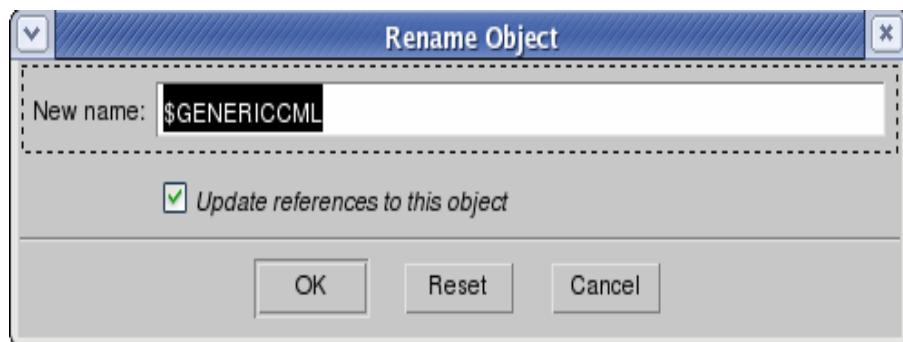
Within the scope of Project Navigator, when you rename a container design object, Pyxis Project Manager reflects the new name of the container by updating the following:

- The container's references.
- The references of any contained objects.
- The references of any objects impacted by the renamed object within the root hierarchy.

Procedure

1. Select the object that needs to be renamed.
2. Bring up the **Rename Object** dialog box, as shown in [Figure 4-23](#). using a menu or the command prompt, in any of the following ways:
 - Command prompt: Type the Rename Object command in the command prompt bar.
 - Pulldown menu: Access the pulldown menu under **Edit > Rename**.
 - Popup menu: Select the **Edit > Rename** option.

Figure 4-23. Rename Object Dialog Box



3. The options have the following meaning:
 - a. **New Name**

Enter the design object's new name in this field.

b. **Update references to this object** (optional)

With this checkbox, you specify whether or not all references are updated to reflect the location of the renamed object. Within the same session, Pyxis Project Manager remembers your selection for this checkbox.

4. Click OK to execute the Rename Object dialog box. The name of the source object is changed to the specified new name.

Related Topics

[Manage Data for a Project](#)

Deleting Design Objects

Deleting a design object within Project Navigator removes all versions of the selected design object. You can delete a single design object or multiple design objects in the same operation.

Procedure

1. Select the object that needs to be deleted.
2. Bring up a deletion confirmation dialog box, as shown in [Figure 4-24](#), using a menu or the command prompt, in any of the following ways:
 - Command prompt: Type the Delete Object command in the command prompt bar.
 - Pulldown menu: Access the pulldown menu under **Edit > Delete > Object**.
 - Popup menu: Select the **Edit > Delete > Object** option.

Figure 4-24. Confirmation of the Delete Object operation



3. The dialog box prompts you to confirm if you want to delete the selected design object.
4. In addition, a checkbox option to “Delete containers with locked objects” is displayed. If you check this option, then even those containers that have locked objects are deleted. By default, this option is not checked.
5. Confirm the deletion by clicking the **Yes** button.

The selected design objects are permanently deleted. You cannot retrieve them.

Related Topics

[Manage Data for a Project](#)

External Libraries and Logic Libraries

External libraries and logic libraries can be included or removed from a project, external or logic library using the **Manage Libraries** menu item, **Edit > External/Logic Libraries**.

In order to include an external library or logic library in a project or another external library, it must use technology settings compatible with those used by the hierarchy in which it is being included.

Note

 Before including the external library or logic library in a project, Pyxis Project Manager validates the process by comparing the default technology selections for the project and the external library.

The Standard MGC Libraries and logic libraries can be included or removed from a project or external library as a group using the Manage Libraries menu item. These libraries can also be managed using the generic external library management functions.

By default, standard libraries are automatically included when using the location map. If you want to navigate to standard libraries, use the Add Standard Libraries button on the Set External Libraries (set_ext_libs) dialog.

Technology libraries are not currently manageable.

Including External and Logic Libraries	349
Removing External and Logic Libraries.....	350
Finding External Dependencies	350

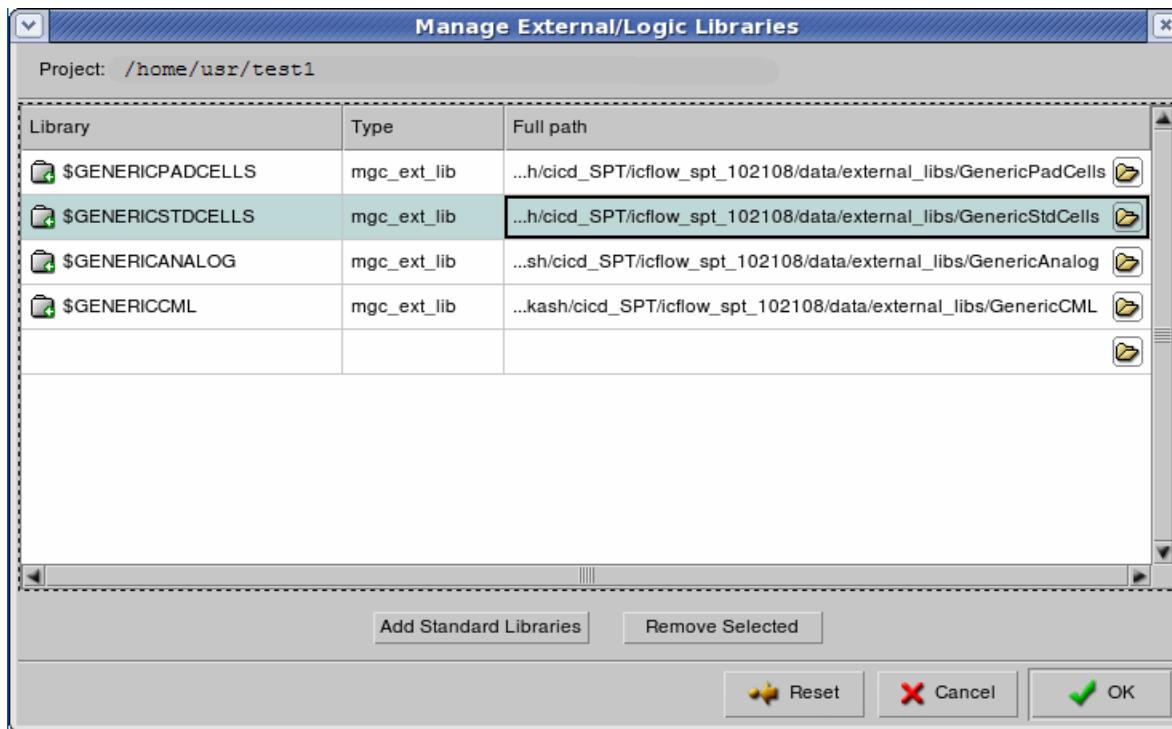
Including External and Logic Libraries

Include external libraries or logic libraries, with a project selected.

Procedure

1. Access the pulldown menu option under **Edit > External/Logic Libraries**.
2. This brings up the Manage External/Logic Libraries dialog box, as shown in [Figure 4-25](#).

Figure 4-25. Manage External/Logic Libraries



3. Select the external or logic libraries that you want to include and click the Add Standard Libraries button.
4. Click the OK button to execute the Manage External/Logic Libraries dialog box.
5. A message is displayed in the Message Area window with a note that the external or logic library reference to the specified path has been included, as shown below.

Note: Included External Library reference to \$GENERICANALOG.

Related Topics

[Removing External and Logic Libraries](#)

[Finding External Dependencies](#)

Removing External and Logic Libraries

Remove an external or logic library, with a project or an external or logic library selected.

Procedure

1. Access the pulldown menu option under **Edit > External/Logic Libraries**.
2. This brings up the Manage External/Logic Libraries dialog box, as shown in [Figure 4-25](#) above.
3. Select the external or logic libraries that you want to remove and click the Remove Selected button.
4. Click the OK button to execute the dialog box.
5. A message is displayed in the Message Area window with a note that the external or logic library reference to the project has been removed, as shown below.

Note: Removed External Library reference to
/scratch1/site/projects/ProjNavDemo/NewLib

Alternatively, you can quickly remove external or logical libraries by using the <delete key> in the Project Navigator.

Related Topics

[Including External and Logic Libraries](#)

[Finding External Dependencies](#)

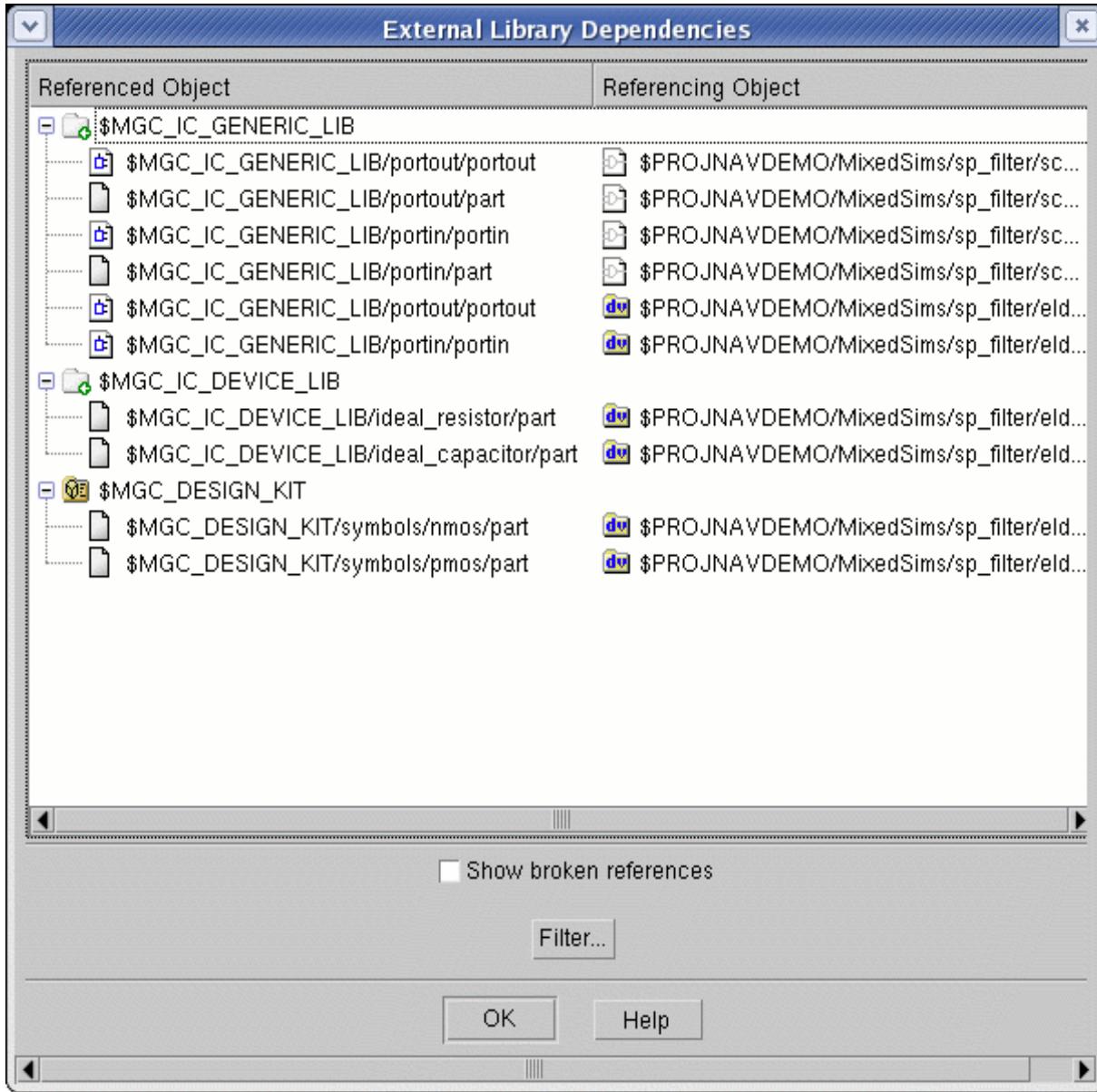
Finding External Dependencies

Obtain a list of all references from the selected design object or hierarchy to objects from other root hierarchical design objects.

Procedure

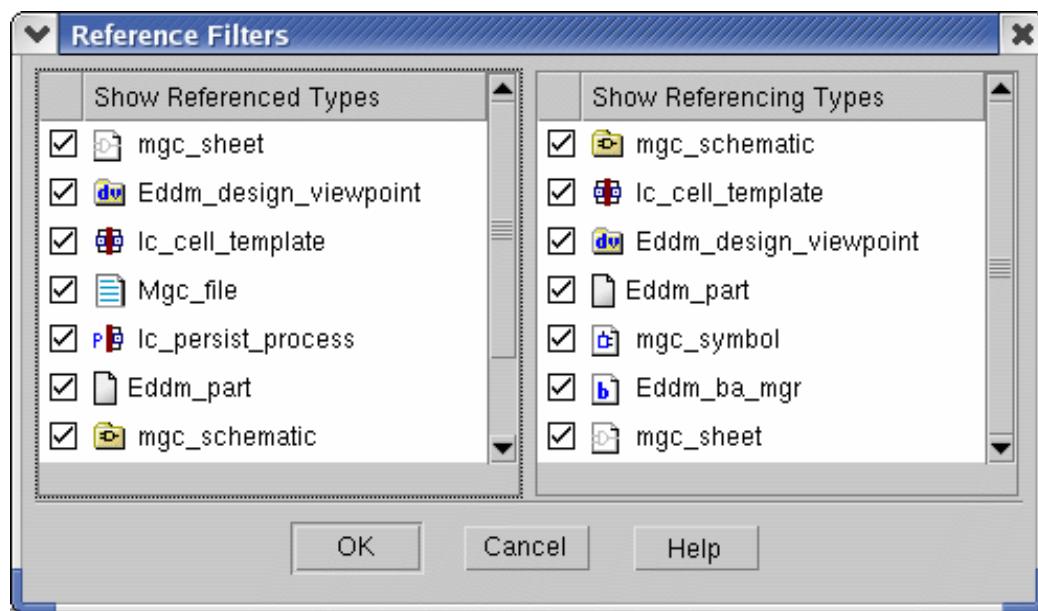
1. Access the pulldown menu option under **Edit > Find External Dependencies**.
2. This brings up the External Library Dependencies dialog box, with a list of referenced and referencing objects, as shown in [Figure 4-26](#).

Figure 4-26. Find External Dependencies Dialog Box



3. Select the **Show broken references** checkbox to list them out. By default, they are not shown in the list.
4. To filter out the object types from the list, click on the Filter button. This brings up the Reference Filters dialog box, as shown in [Figure 4-27](#),

Figure 4-27. Reference Filters Dialog Box



5. In the Reference Filters dialog box, check or uncheck the objects that you want displayed in both the Show Referenced Types and Show Referencing Types lists.
6. Once you have filtered the references, click OK to execute the Reference Filters dialog box.
7. Click OK to exit the External Library Dependencies dialog box.

Related Topics

[Removing External and Logic Libraries](#)

[Including External and Logic Libraries](#)

Manage Technology Settings

Once a project or external library is created, you can set, validate or view the technology settings using the **Edit > Technology** pulldown menu options.

Changing the Technology Settings	352
Validating the Technology Settings	353
Viewing the Technology Settings	354

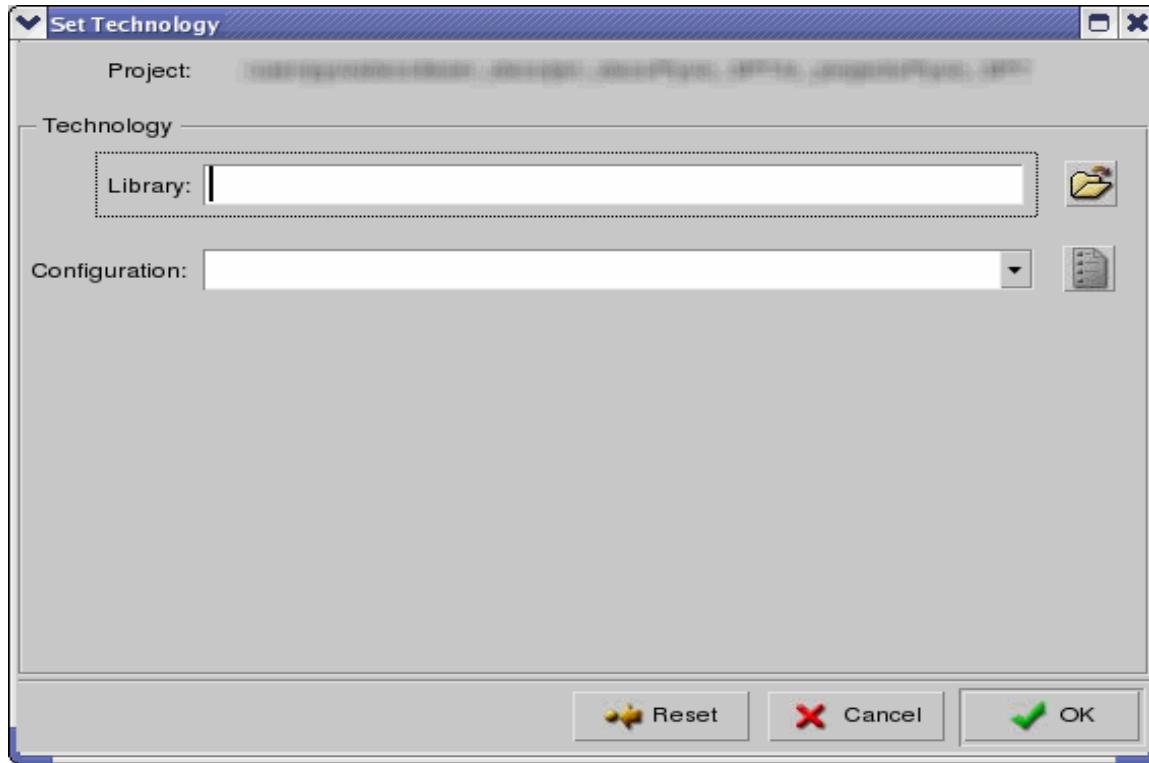
Changing the Technology Settings

Change the technology setting for the selected project or external library.

Procedure

1. Access the pulldown menu using the **Edit > Technology > Set** option, which brings up the Set Technology dialog box, as shown in [Figure 4-28](#).

Figure 4-28. Set Technology Dialog Box



2. Specify the technology library and technology configuration that make up the technology set for the selected project or external library.
3. Click the OK button to execute the Set Technology dialog box.
4. A message is displayed in the Message Area and Transcript Area windows, with a note that it has refreshed all the open hierarchies with your new technology settings.

Note: Refreshing all open hierarchies.

Note: Refresh complete!

Related Topics

[Validating the Technology Settings](#)

[Viewing the Technology Settings](#)

Validating the Technology Settings

Verify that all the external libraries included within the selected project or external library use compatible technology settings.

Procedure

1. Access the pulldown menu using the **Edit > Technology > Validate** option,
2. A message is displayed in the Message Area and Transcript Area, with a note that it has either validated the technology or that the technology has errors, as shown below.

When the technology settings are valid, you get a message similar to the following:

Note: Technology for the hierarchy
/scratch1/site/projects/ProjNavDemo/NewLib is valid. See monitor for details.

When the technology settings are invalid, you get a message similar to the following:

Note: Technology for the hierarchy
/scratch1/site/projects/ProjNavDemo/NewLib is invalid. See monitor for details.

Note

 Press the F8 function key to view the Monitor window for more information.

Related Topics

[Changing the Technology Settings](#)

[Viewing the Technology Settings](#)

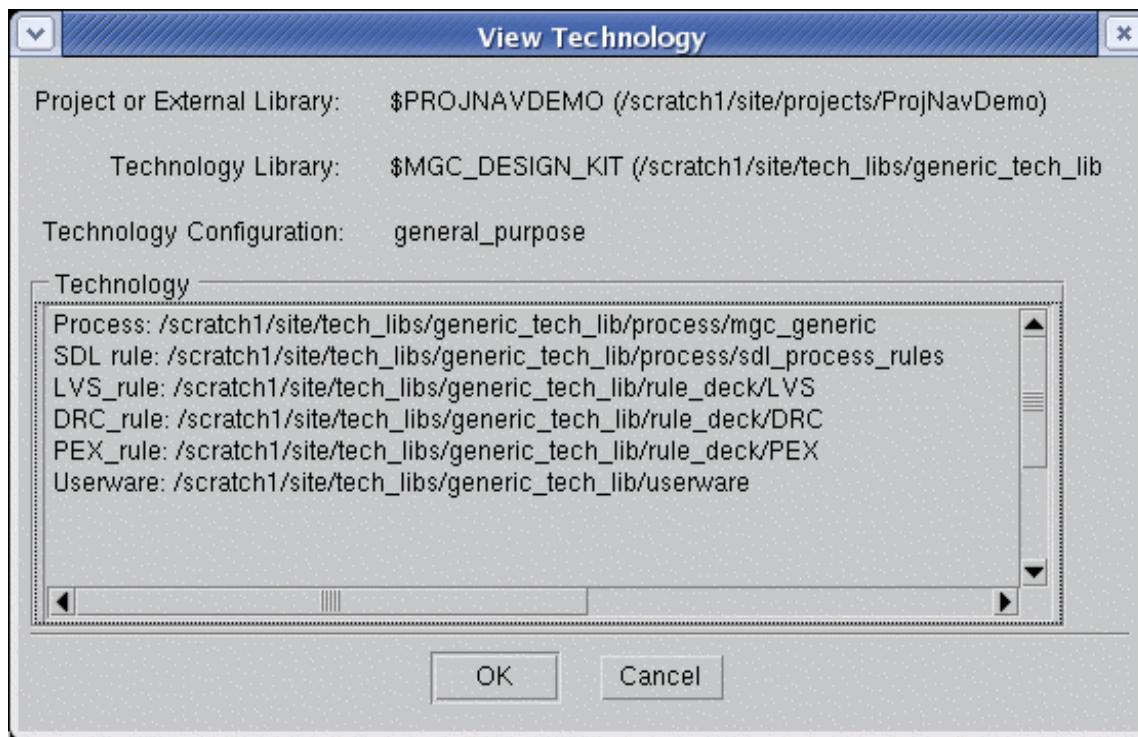
Viewing the Technology Settings

View the current technology settings for the selected project or external library.

Procedure

1. Access the pulldown menu using the **Edit > Technology > View** option,
2. The View Technology dialog box appears, as shown in [Figure 4-29](#).

Figure 4-29. View Technology Dialog Box



3. The View Technology dialog box displays the details of the project or external library, and its technology settings (i.e. the technology library and the technology configuration).
4. Click the OK button to close the View Technology dialog box.

Related Topics

[Validating the Technology Settings](#)

[Changing the Technology Settings](#)

Chapter 5

Revision Control

By default, revision control is not turned on in the software. You can enable revision control by setting the appropriate environment variables.

 **Note** Both Subversion (SVN) and ClioSoft (SOS) revision control systems can be used in Pyxis Project Manager.

Revision Control Terminology	357
User Credentials	359
Revision Control Environment Variables and Setup	359
Object Status States	363
Revision Control Menu in Pyxis Project Manager	371
ClioSoft Revision Control Procedures	374
Subversion Revision Control Procedures	385
Common Revision Control Procedures	390
Subversion Administrator Information	398
Integrating Custom Type Data with Revision Control	403
Qualification Scripts and Revision Control Integration	403
ClioSoft Troubleshooting	405

Revision Control Terminology

The following terminology applies to both ClioSoft and Subversion revision control.

- **HDO** — A hierarchical design object.
- **Managed** — An object is managed if it is in a managed work area and has been checked into the repository. A work area is managed if it was created using the PMPN command ‘Create Work Area or Add Project.’
- **Project Manager PN** — (Pyxis) Project Manager, Project Navigator. The new window in Pyxis Project Manager used to view design data which is organized into projects and libraries.
- **PMPN** — (Pyxis) Project Manager, Project Navigator.

- **Repository** — Path to the storage location on the revision control server where a particular project can be found.
- **Revision** — The status of an object at a given point in time in the repository. Also known as version.
- **Revision Control System (RCS)** — The version control system or management of Project Manager projects.
- **Server** — URL to the machine resource used to supply revision control services.
- **SOS** — ClioSoft's tool. Please refer to ClioSoft's documentation for additional information.
- **SVN** — Subversion's tool. Please refer to Subversion documentation for additional information.
- **Update** — Sync up changes made in the repository with the work area.
- **Work Area** — The local copy of objects from a repository with a specific time or revision.

Related Topics

[Creating a Work Area \(SOS\)](#)

[Creating a Work Area \(SVN\)](#)

[Object Status States](#)

[Adding a Project to a Repository \(SOS\)](#)

[Adding a Project to a Repository \(SVN\)](#)

[Revision Control Environment Variables and Setup](#)

User Credentials

User credentials and environment variables are specific to Subversion.

Subversion

Subversion requires a username and password for some operations. This only needs to be provided once per server. Run a command requiring authentication to achieve credentials. This method of establishing authentication credentials applies to the Apache server and to *svnserve* (the *svnserve* case would use URLs beginning with *svn://myserver*).

Example 5-1. Establishing Credentials

Entering the following into a terminal window:

```
svn info http://myserver/svn/my_repo/
```

Displays the following:

```
Authentication realm: <http://myserver:80> Subversion repository
Password for 'UNIX_USERNAME':
```

Press **Enter** to type username and password:

```
Username: bob
Password for 'bob': pass1234
```

This process creates a file containing your credentials within your home directory (`~/.subversion/auth/svn.simple/`). The filename is something like `1de1c2a32049a86020946ef5e3ebfb20`.

To use another username and password, you can either use a different Unix account or delete this file to use the same Unix account.

Related Topics

[Subversion Revision Control Procedures](#)

Revision Control Environment Variables and Setup

There are certain environment variables and tasks required to setup your revision control environment in Pyxis Project Manager.

Setting Up the ClioSoft Environment	360
Setting Up the Subversion Environment	362
Pyxis Project Setup	362

Environment Variables

The following environment variables are required to run Pyxis Project Manager and other point tools with revision control.

ClioSoft Environment Variables

- AMPLE_PATH - points to \$CLIOSOFT_DIR/adaptors/dmgr/userware.
- CLIOSOFT_DIR - points to a ClioSoft install.
- MGC_SOS_SERVER - points to the name of your SOS server. Optional.
- MGC_SOS_PROJECT - points to the name of your SOS project. Optional.

Subversion Environment Variables

- AMPLE_PATH - points to \$MGC_HOME/shared/examples/rcs/subversion.
- MGC SVN REPOSITORY - points to your SVN server. This location could be a URL.

Common Environment Variables

- MGC_RCS_LOGFILE - points to the location of the revision control log file. This defaults to \$MGC_HOME/tmp if not set.

Example 5-2. Subversion Revision Control Variables

```
$MGC_SVN_REPOSITORY=http://myserver/svn/my_repo/trunk
$AMPLE_PATH=$MGC_HOME/shared/examples/rcs/subversion
$MGC_RCS_LOGFILE=$MGC_HOME/tmp
```

Related Topics

[Setting Up the ClioSoft Environment](#)

[Setting Up the Subversion Environment](#)

Setting Up the ClioSoft Environment

Set up your environment to use ClioSoft with Pyxis.

Prerequisites

- Setup your environment to use the appropriate Pyxis installation and ClioSoft installation. For the Pyxis v10.5_1 release, use ClioSoft SOS 6.32.p2.
- Confirm that MGC_HOME and CLIOSOFT_DIR are properly set.

Procedure

1. Create and start an SOS server and project. For more information, please refer to ClioSoft's documentation.
2. Set CLIOSOFT_DIR to the ClioSoft install.
3. Set AMPLE_PATH to `$CLIOSOFT_DIR/adaptors/dmgr/userware`.
4. Optional:
 - Set MGC_SOS_SERVER to the name of your SOS server.
 - Set MGC_SOS_PROJECT to the name of your SOS project.
5. Optional: To confirm what version of SOS is running select **Revision Control > About Revision Control**.

Results

Your Pyxis environment is now setup properly to use ClioSoft.

Related Topics

[ClioSoft Revision Control Procedures](#)

[Pyxis Project Setup](#)

Setting Up the Subversion Environment

Set up your environment to use Subversion with Pyxis.

Prerequisites

- Setup your environment to use the appropriate Pyxis installation and Subversion installation. For the Pyxis v10.5_1 release, use SVN 1.4.6.
- Confirm that MGC_HOME is properly set.

Procedure

1. Create and start an SVN repository. See [Creating a Repository](#) for more information.
2. Set MGC SVN REPOSITORY to your SVN repository.
3. Set AMPLER_PATH to *\$MGC_HOME/shared/examples/rcs/subversion*.
4. Optional:
 - Set MGC RCS LOGFILE to a location where the revision control system saves a log file. By default, this location is *\$MGC_HOME/tmp*.
5. Optional: To confirm what version of SVN is running select **Revision Control > About Revision Control**.

Results

Your Pyxis environment is now setup properly to use Subversion.

Related Topics

[Subversion Revision Control Procedures](#)

[Pyxis Project Setup](#)

Pyxis Project Setup

When a Pyxis PMPN project is created, it includes a location map inside the root container.

For instance:

```
MGC_LOCATION_MAP_3
$SIMPLE -t LIBRARY
. .
INCLUDE $MGC_HOME/shared/etc/app1/base/mgc_map_template
INCLUDE /net/depot/design_kits/simple_rf/mgc_tech_location_map
```

Note the include for */net/depot/design_kits/simple_rf/mgc_tech_location_map*. For work areas to function correctly, this included path needs to be in a common location.

Alternatively, the path can use an environment variable such as \$DESIGN_KITS.

```
INCLUDE $DESIGN_KITS/simple_rf/mgc_tech_location_map
```

You would then set this environment variable to point to the location of your designs.

Related Topics

[Setting Up the ClioSoft Environment](#)

[Setting Up the Subversion Environment](#)

Object Status States

Object status describes the result of a comparison between your local copy versus the latest version in the repository.

Figure 5-1 below shows the various states and their icons. It also describes the order of precedence for those states.

For example, an object that is both **Modified** and **Locked by Me** displays as **Modified** because it is higher in precedence. Table 5-1 below shows descriptions of each state.

Caution

 Not all status states are supported by each revision control system. For example, ClioSoft does not support Obstructed, Repository Added, Repository Deleted, or Missing.

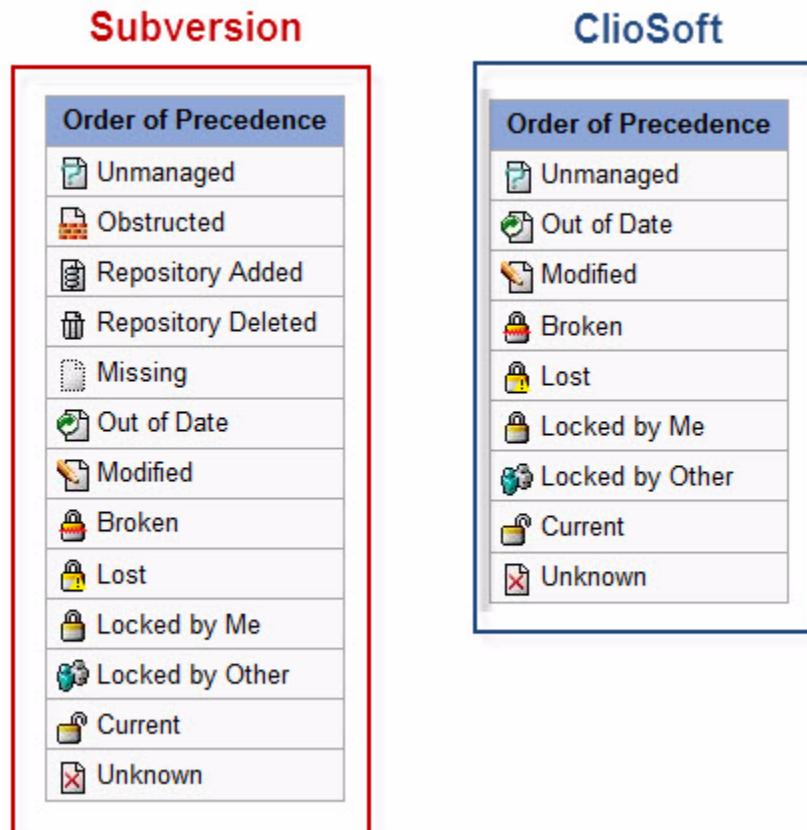
Table 5-1. Object Status Descriptions

Object Status	Description
Unmanaged	The local object is not under revision control.
Obstructed	The local object is unmanaged; however, the repository already has an object in its place. Options include moving, renaming, or deleting the local object. Subversion only.
Repository Added	There is a new object in the repository that is not yet in the work area. Subversion only.
Repository Deleted	The local object is now unmanaged because the object in the repository was deleted. Updating this object results in the local object being deleted. Subversion only.
Missing	The object is missing from the work area when it should be present. Subversion only.
Out of Date	A newer version exists in the repository that can be retrieved by updating.

Table 5-1. Object Status Descriptions (cont.)

Object Status	Description
Modified	The local object has edits. A checkin uploads your changes to the repository.
Broken	The object is checked out by you but is no longer locked by you in the repository. Try checkout to restore the lock or cancel checkout to unlock the object in this work area.
Lost	The object is checked out by you but not in this work area. Change to the work area where the local originated or try cancel checkout as admin.
Locked by Me	The local object is checked out by you. Use cancel checkout to unlock it.
Locked by Other	The object is checked out by someone else. No check ins until that user unlocks the object.
Current	The local object and its repository version are the same.
Unknown	The status is unknown. An error unexpectedly occurred.

Figure 5-1. Object Status - Order of Precedence



Object Status Flow	365
Viewing Object Status	366
Filtering Objects	369

Object Status Flow

The object status follows a particular flow in the repository.

Figure 5-2 illustrates the flow with text descriptions of possible results listed below.

Figure 5-2. Object Status Flow

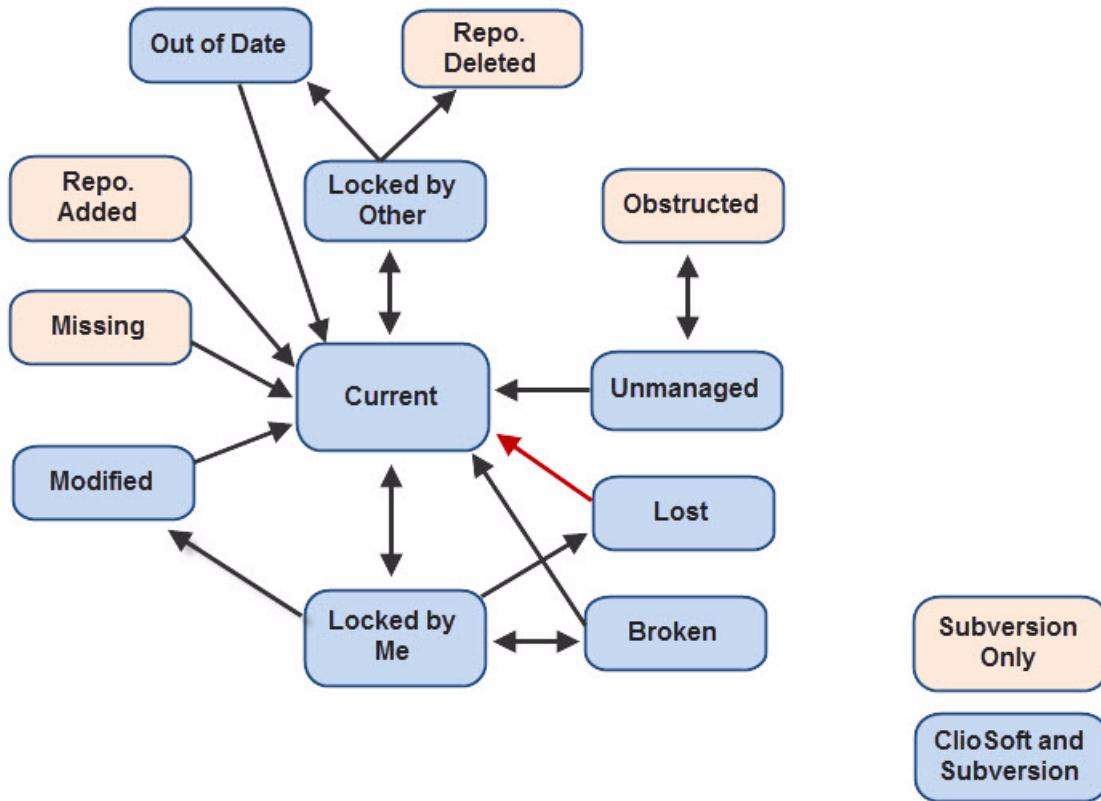


Table 5-2. Starting and Ending Object Status

Starting Object Status

Unmanaged objects can become . . .

Current objects can become . . .

Ending Object Status

- **Current** when you perform check in.
- **Obstructed** when another user performs check in of a duplicate object.

- **Locked By Me** when you perform checkout.
- **Locked by Other** when someone else performs checkout.

Starting Object Status

Out of Date objects can become . . .

Modified objects can become . . .

Locked by Me objects can become . . .

Locked by Other objects can become . . .

Broken objects can become . . .

Lost objects can become . . .

Repository Added objects can become . . .

Missing objects can become . . .

Ending Object Status

- **Current** when you perform an update.
- **Current** when you perform check in or cancel checkout.
- **Modified** when you perform edits.
- **Current** when you perform cancel check out.
- **Broken** when another user performs admin cancel check out.
- **Lost** when you switch work areas.
- **Repository Deleted** when another user performs delete.
- **Out of Date** when another user performs check in.
- **Current** when another user performs cancel checkout.
- **Current** when you perform cancel checkout.
- **Current** when you perform admin cancel checkout. Highlighted by red arrow in [Figure 5-2](#).
- **Current** when you perform an update.
- **Current** when you perform an update or cancel a checkout.

Related Topics

[Viewing Object Status](#)

[Filtering Objects](#)

Viewing Object Status

View the status of an individual object or group of objects.

You may also perform revision control operations from the dialog box including: checkout, checkin, cancel checkout, and update. There is a refresh option to update the table with the latest object status and all operations support a recurse flag.

Prerequisites

- You must have permission to view the object (Subversion only).
- The revision control system must be running.

Procedure

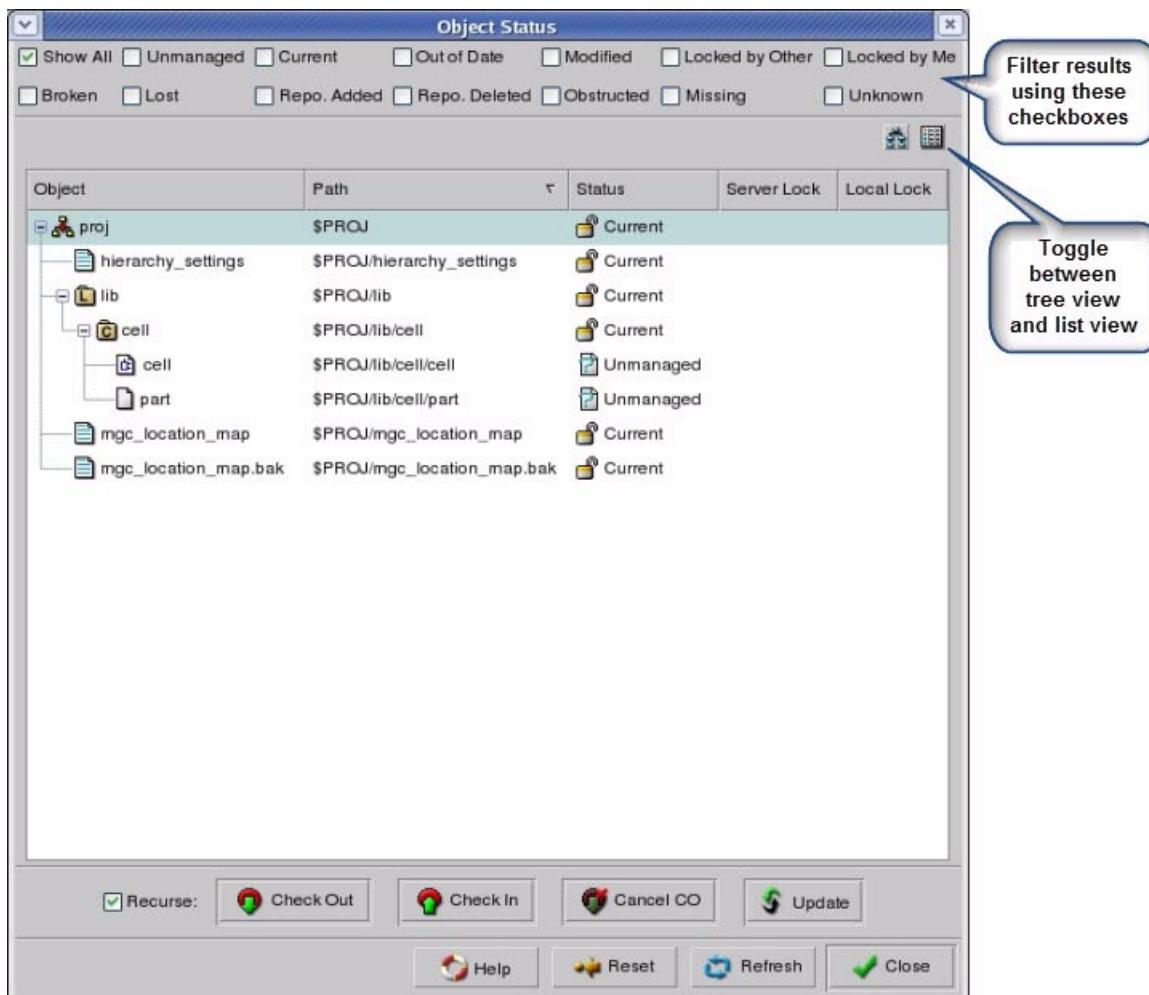
1. Select **Revision Control > Status**.

The Revision Control System Object Status dialog box appears as shown in [Figure 5-3](#) below using a tree view. You can toggle the display between a tree view and a list view. This dialog box contains a sortable tree of objects.

2. Right-click to display a pop-up menu. This menu enables you to collapse and expand items.
3. Filter objects by checking one or more check boxes located at the top of the dialog box. Objects that do not match the filter are grayed out or removed from the display.

Results

Figure 5-3. Object Status (Tree View) Dialog Box



The object status states are filtered and displayed in the dialog box.

Related Topics

[Object Status Flow](#)

[Object Status States](#)

Filtering Objects

Set up revision control filter options.

Prerequisites

- You must have permission to view the object (Subversion only).
- The revision control system must be running.

Procedure

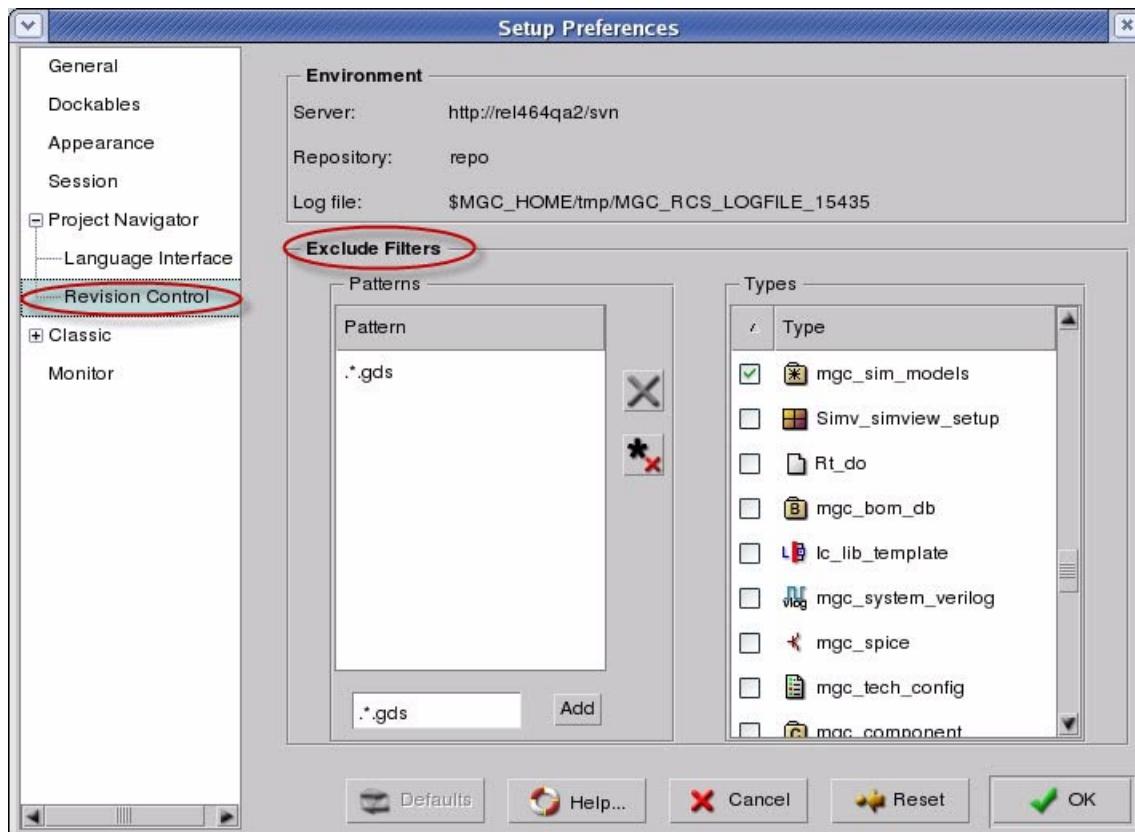
1. Select **Setup > Preferences > Project Navigator > Revision Control**.

The Setup Preferences dialog box displays as shown in [Figure 5-4](#). This dialog box displays the filter options stored for the current user.

2. Select objects to ignore during revision control operations by pattern or by type in the **Exclude Filters** section, and Select **Apply**.

Results

Figure 5-4. Setup Preferences Revision Control Dialog Box



The selected filter exclusions are applied. In this example, a filter is applied to the pattern “.*.gds”. This filter is used during RCS operations to ignore all files matching this pattern.

Revision Control Menu in Pyxis Project Manager

To access: Revision Control menu.

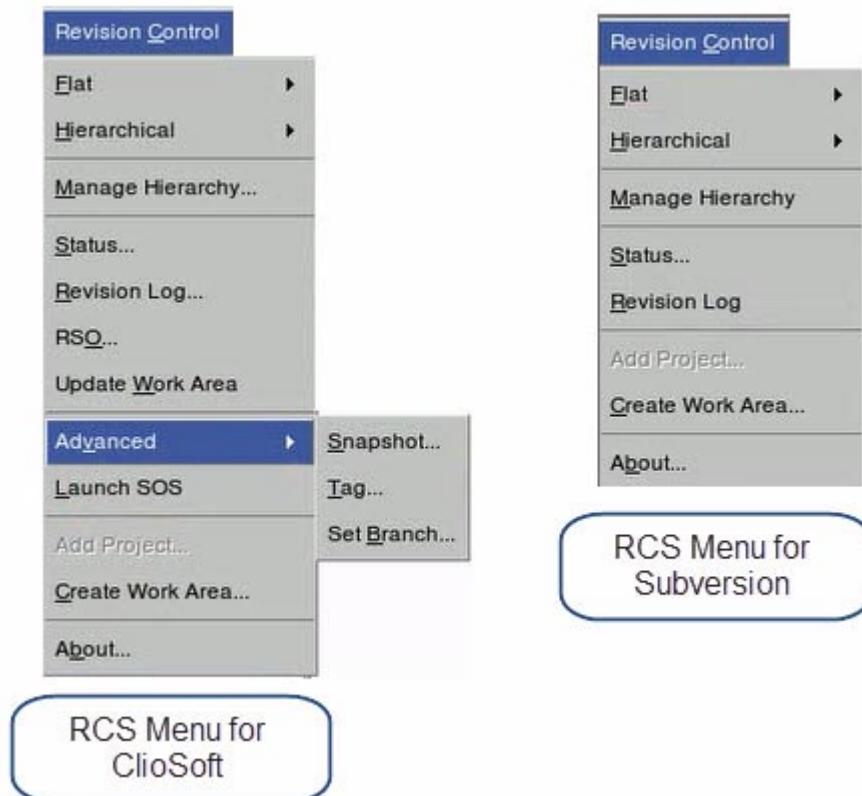
The revision control operations are found within the Revision Control menu.

Description

This menu, shown in [Figure 5-5](#), is only displayed when revision control is enabled by setting the appropriate environment variables and the menu varies depending on whether you are using ClioSoft or Subversion. Common RCS operations are available via the right mouse button pop-up.

There are different environment variables required depending on whether you are using Subversion or ClioSoft. The menu items found under the **Flat** menu section operate on the selected objects alone and the subdirectories are ignored. The items under the **Hierarchical** menu section operate on the selected objects along with all the objects inside any subdirectories.

Figure 5-5. Revision Control Menu



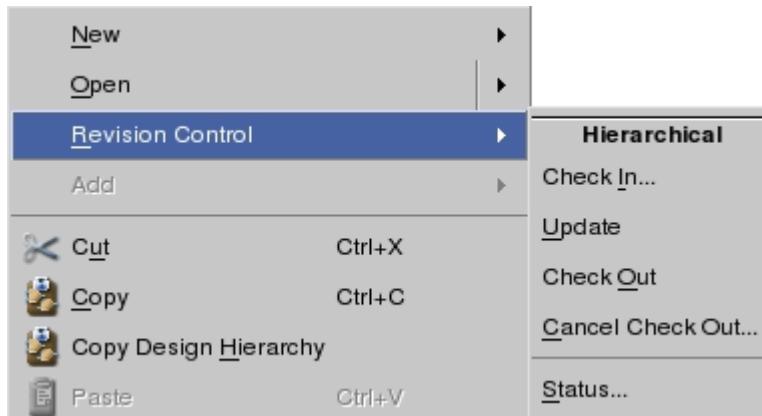
Fields

Table 5-3. Revision Control Menu Items

Item	Description	Note
Check Out	If you have a valid managed work area, you can ‘check out’ objects in that work area. This locks the object in the repository so that other users cannot make changes to the object or to check changes in.	Flat Hierarchical See Checking Out Objects .
Cancel Check Out	Cancels the check out action and unlocks the object.	Flat Hierarchical See Canceling an Object Check Out .
Check In	Commit changes made to an object in a local work area to the repository and unlocks the object.	Flat Hierarchical See Checking In Objects and Using Automatic Check In .
Update	Sync up changes made in the repository with the work area.	Flat Hierarchical See Updating an Object .
Status	View a particular object’s status in the repository.	See Viewing Object Status .
Revision Log	Display all the revisions for a selected object or to revert a revision.	See Changing the Revision Search Order (RSO) or Using the Revision Log (SVN) .
Add Project	Add a project to the repository.	See Adding a Project to a Repository (SOS) .
Create Work Area	A directory maintaining a <i>copy</i> of a revision controlled project. Objects in the work area are managed. They can only be edited if they are ‘checked out.’	See Creating a Work Area (SOS) .

You can access common revision control operations by right-clicking on an object as shown in Figure 5-6.

Figure 5-6. Revision Control Pop Up Menu



Related Topics

[Setting Up the ClioSoft Environment](#)
[ClioSoft Revision Control Procedures](#)
[Common Revision Control Procedures](#)

[Setting Up the Subversion Environment](#)
[Subversion Revision Control Procedures](#)

ClioSoft Revision Control Procedures

There are certain tasks associated with using ClioSoft revision control in Pyxis Project Manager.

Adding a Project to a Repository (SOS)	374
Creating a Work Area (SOS)	376
Changing the Revision Search Order (RSO)	377
Using the Revision Log (SOS)	379
Applying Snapshots	381
Tagging Objects	382
Setting a Branch	383

Adding a Project to a Repository (SOS)

Publish or add a project to your repository. The Add Project dialog box displays all available SOS servers including their SOS projects.

Prior to populating the repository, you must decide the structure of that project, following the constraints described in this document. You need to invoke Pyxis Project Manager PN in a staging area and construct a project, add libraries, modify location maps, and add references to a technology library and external libraries. You may also import older data into a Pyxis Project Manager Project staging area.

The available root HDO types that you can place under revision control directly are:

- Project (mgc_project)
- External Library (mgc_ext_lib)
- Logic Library (mgc_logic_lib)
- Technology Library (mgc_tech_lib)

Prerequisites

- You must have permissions to add new objects to the revision control system.
- The revision control system must be running at the time you populate the repository.

Procedure

1. Select a Pyxis project or external, logic, or technology library to add.
2. Select **Revision Control > Add Project**.

The Add Project Dialog Box displays as shown in [Figure 5-7](#).

3. Select the SOS Server and SOS Project from the available lists.

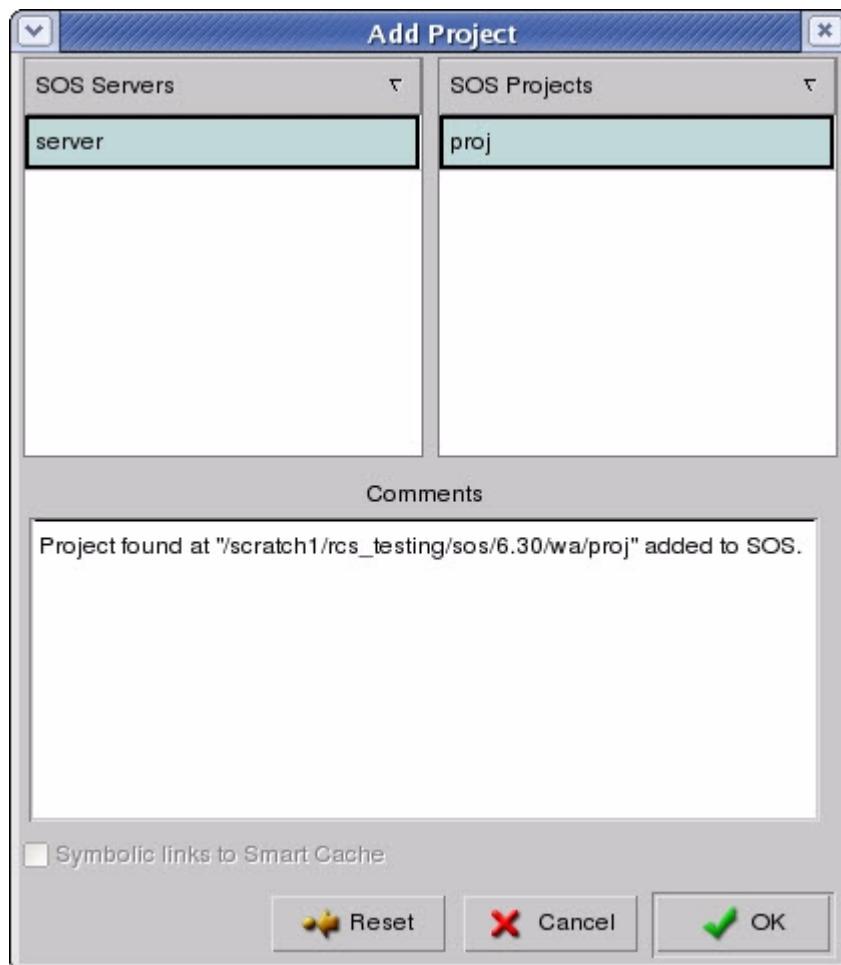
4. Optional: Type a comment into the **Comments** field.

Note: The “Symbolic links to Smart Cache” checkbox is automatically checked if the specified SOS server/project is enabled with this feature. Otherwise, the checkbox is unchecked and disabled.

5. Select the **OK** button to add the project to the repository.

Results

Figure 5-7. Add Project Dialog Box (SOS)



A project is added to the repository.

Related Topics

[ClioSoft Revision Control Procedures](#)

[\\$add_project_to_rc\(\)](#)

Creating a Work Area (SOS)

Begin work on a project which is managed in a revision control system.

This dialog box displays all SOS servers including their SOS projects. The RSO can be changed prior to creating a work area.

The **Symbolic links to Smart Cache** checkbox is available if the SOS server/project is enabled with this feature.

Prerequisites

- You must have permissions to update a work area from the revision control system.
- The revision control system must be running at the time of work area creation.

Procedure

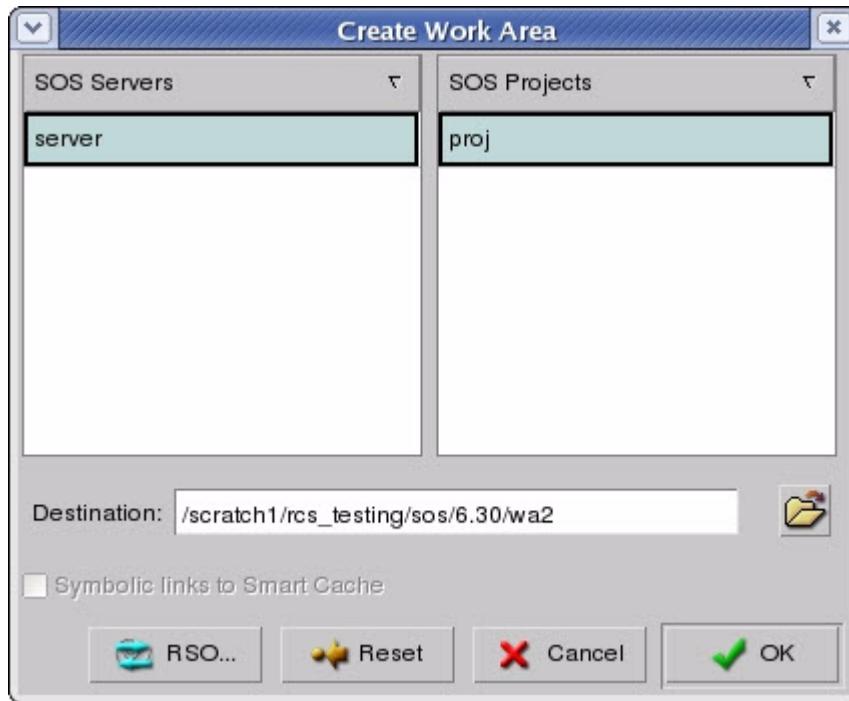
1. Select **Revision Control > Create Work Area**.

The Create Work Area Dialog Box displays as shown in [Figure 5-8](#).

2. Select an SOS server and project from the available list.
3. Specify a destination path for the work area to be created using the **Browse**  button or by typing the path in the text entry field.
4. Select **OK** to create the work area.

Results

Figure 5-8. Create Work Area Dialog Box (SOS)



A work area is created.

Related Topics

[ClioSoft Revision Control Procedures](#)

[\\$create_work_area_from_rc\(\)](#)

[Changing the Revision Search Order \(RSO\)](#)

Changing the Revision Search Order (RSO)

Change the RSO by adding or removing labels.

Changes made in this dialog box persist for future operations such as create work area and update as well as for user sessions.

Prerequisites

- You must have permissions to update a work area from the revision control system.
- The revision control system must be running at the time of work area creation.

Procedure

1. Select **Revision Control > RSO**.

The RSO dialog box displays as shown in [Figure 5-9](#).

2. Choose an existing label from the list or enter a new one.
3. Select the **Add** button to move the selected items in the **RSO** section.
4. Change the RSO labels, as needed.
5. Use the up and down arrow keys to change the order of the displayed items.
6. Selected the **OK** button.

Results

Figure 5-9. RSO Dialog Box



The revision search order is updated for the selected items.

Related Topics

[ClioSoft Revision Control Procedures](#)

[Creating a Work Area \(SOS\)](#)

Using the Revision Log (SOS)

Display all the revisions for a selected object or to revert (that is, back out) a revision.

The remaining sections of the dialog box populate once you select a revision.

Prerequisites

- You must have permissions to update a work area from the revision control system.
- The revision control system must be running at the time of work area creation.

Procedure

1. Select a view that is not modified or checked out.

2. Select **Revision Control > Revision Log**.

Repository versions are listed with the author, date, and comment in the Revision Log dialog box as shown in [Figure 5-10](#).

3. Select an older version from the list in the Revision Log dialog box. The selection is highlighted.

4. Select the “Do not change repository (local read-only)” check box.

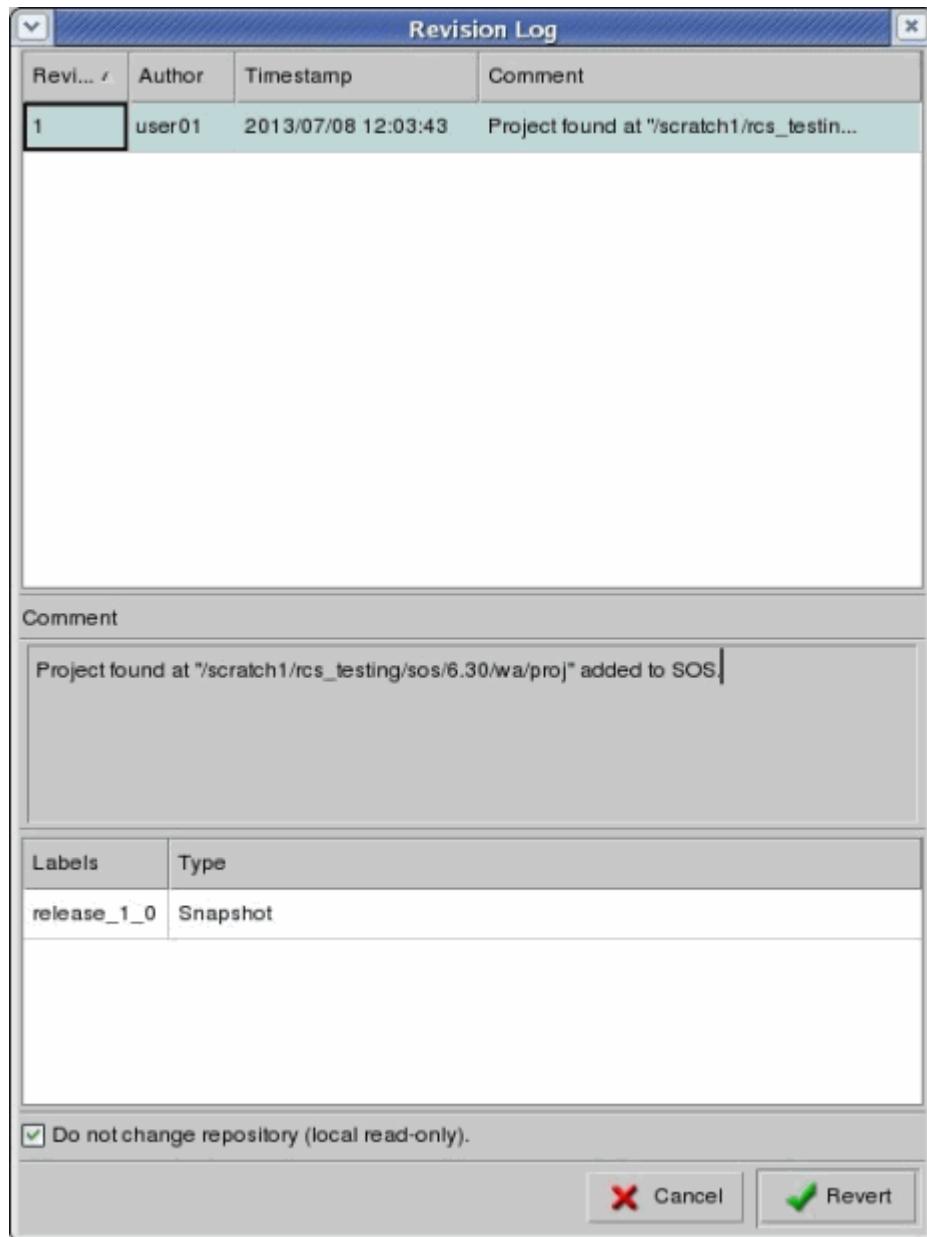
5. Select the **Revert** button.

The selected revision is reverted and a comment is added automatically.

6. Uncheck the “Do not change repository (local read-only)” check box.

Results

Figure 5-10. Revision Log Dialog Box (SOS)



The selected revision is reverted.

Related Topics

[ClioSoft Revision Control Procedures](#)

Applying Snapshots

Apply a new or existing snapshot to an object. To use an existing snapshot requires administrator privileges.

Prerequisites

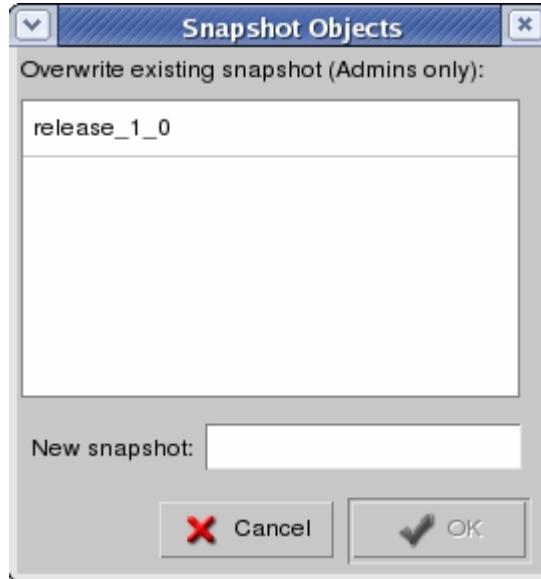
- You must have permissions to update a work area from the revision control system.
- The revision control system must be running at the time of work area creation.

Procedure

1. Select **Revision Control > Advance > Snapshot**.
2. Choose an object from the available list.
3. Type a name in the **New snapshot** field as shown in [Figure 5-11](#).
4. Select the **OK** button.

Results

Figure 5-11. Snapshot Objects Dialog Box



You have applied a new snapshot for the selected object(s).

Related Topics

[ClioSoft Revision Control Procedures](#)

Tagging Objects

Apply a new or existing tag to objects. You can remove a tag (also known as Back Off) from objects.

Prerequisites

- You must have permissions to update a work area from the revision control system.
- The revision control system must be running at the time of work area creation.

Procedure

1. Select **Revision Control > Advance > Tag**.
2. Select the **Tag** radio button and select an existing tag from the available list or enter a new tag as shown in [Figure 5-12](#).

Type a name in the **Apply new tag** field if you want to apply a new tag name rather than an existing one.

If you want to remove a tag rather than applying one, select the **Back Off** radio button.

3. Select the **OK** button.

Results

Figure 5-12. Tag Objects Dialog Box



You have either applied a new or existing tag to objects or you have removed a tag from objects.

Related Topics

[ClioSoft Revision Control Procedures](#)

Setting a Branch

Set a new or existing branch.

The default branch is indicated by the word “default” next to the branch name. The **Branch project** option determines whether the whole SOS project is branched or just the selected objects are branched.

Prerequisites

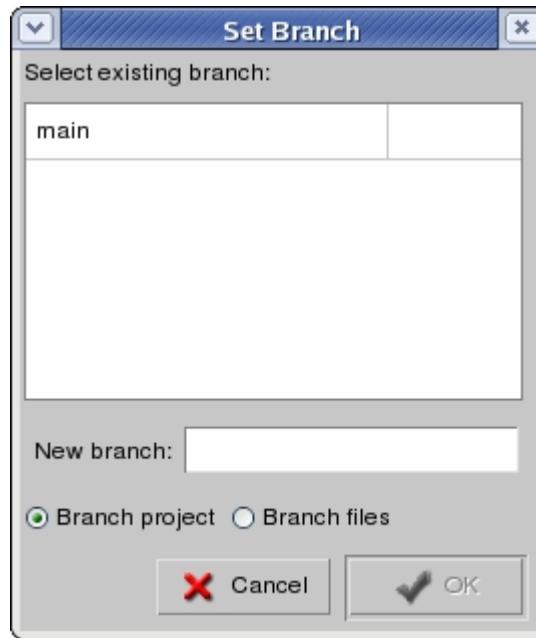
- You must have permissions to update a work area from the revision control system.
- The revision control system must be running at the time of work area creation.

Procedure

1. Select **Revision Control > Advance > Set Branch**.
2. Choose an existing branch.
3. Type a name into the **New branch** field.
4. Specify **Branch project** or **Branch files** as shown in [Figure 5-13](#).
5. Select the **OK** button.

Results

Figure 5-13. Set Branch Dialog Box



You have set a new or existing branch.

Related Topics

[ClioSoft Revision Control Procedures](#)

Subversion Revision Control Procedures

There are certain tasks associated with using Subversion revision control in Pyxis Project Manager.

Adding a Project to a Repository (SVN)	385
Creating a Work Area (SVN)	386
Using the Revision Log (SVN).....	387

Adding a Project to a Repository (SVN)

Publish or add a project to your SVN repository.

The **Add Project** dialog box, as shown in [Figure 5-14](#), is available when an unmanaged project is selected. The active SVN repository is displayed in a hierarchy browser for selecting a destination.

Prerequisites

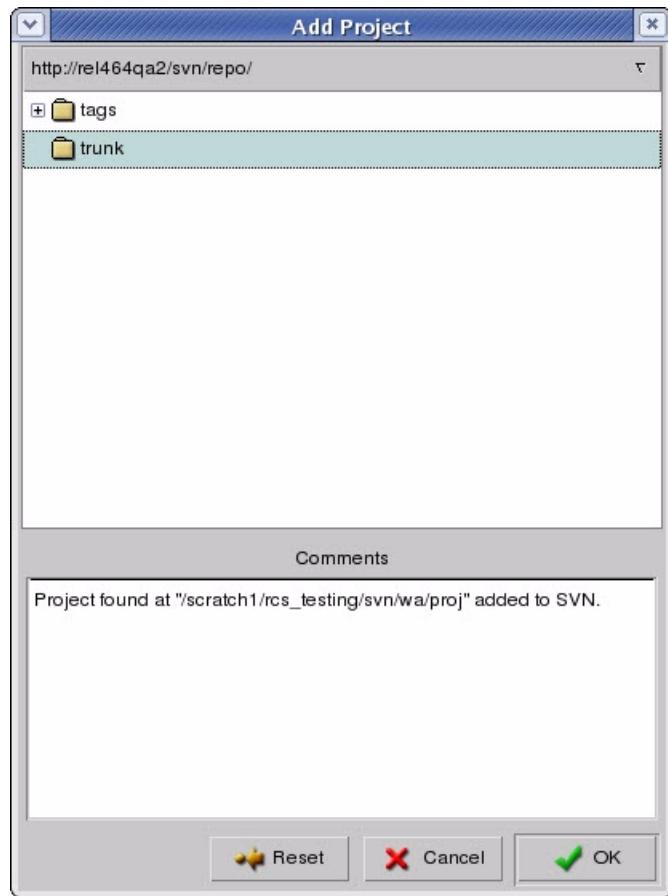
- You must have permissions to update a work area from the revision control system.
- The revision control system must be running at the time of work area creation.

Procedure

1. Select **Revision Control > Add Project**.
2. Expand the hierarchy listing and select the appropriate destination.
3. Optional: Add specific comments or use the default comment.
4. Select the **OK** button.

Results

Figure 5-14. Add Project Dialog Box (SVN)



A project has been added to your repository.

Related Topics

[\\$add_project_to_rc\(\)](#)

[Subversion Revision Control Procedures](#)

Creating a Work Area (SVN)

Begin work on a project which is managed in a revision control system.

The **Create Work Area** dialog box, as shown in [Figure 5-15](#), displays the active SVN repository in a hierarchy browser for selecting Project Manager data.

Prerequisites

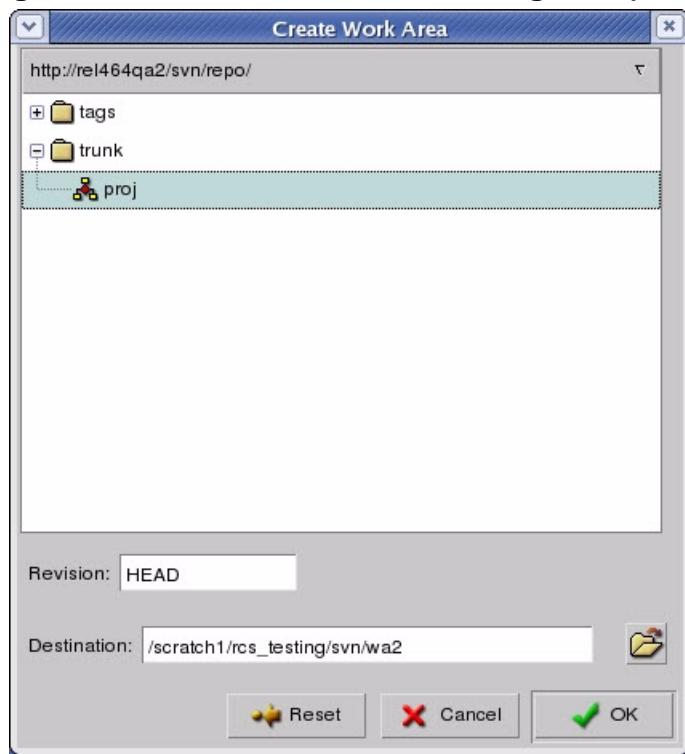
- You must have permissions to update a work area from the revision control system.
- The revision control system must be running at the time of work area creation.

Procedure

1. Select **Revision Control > Create Work Area**.
2. Expand the hierarchy browser and choose the appropriate project.
3. Specify a destination by using the **Browse** button or by typing a location in the **Destination** field.
4. Optional: A revision can be specified in the **Revision** field. The default value is “HEAD,” which is the latest revision.

Results

Figure 5-15. Create Work Area Dialog Box (SVN)



Related Topics

[Subversion Revision Control Procedures](#)

[\\$create_work_area_from_rc\(\)](#)

Using the Revision Log (SVN)

Display all the revisions for a selected object or to revert a revision.

A revert can be applied to the following situations:

- the whole change set for a revision when no objects are selected under **Changes**.

- to individual objects within a change set when any object is selected under **Changes**.
- to the entire work area. The **Rollback Work Area** button enables the entire work area to appear as it did for the selected revision.

There are two options available to use revert:

- **Use revision**
 - **Before** radio button uses the revision prior to the selected revision to revert. This means that the changes made for revision N are not in the work area. Use this option in situations where you want to undo changes made for a revision such as a delete.
 - **After** radio button uses the selected revision to revert. The changes made for revision N are in the work area.
- **Edits**
 - **Read-only** radio button disallows check-ins by making objects out-of-date.
 - **Writable** radio button allows the reverted changes to be checked in to complete the revert within the repository.

Prerequisites

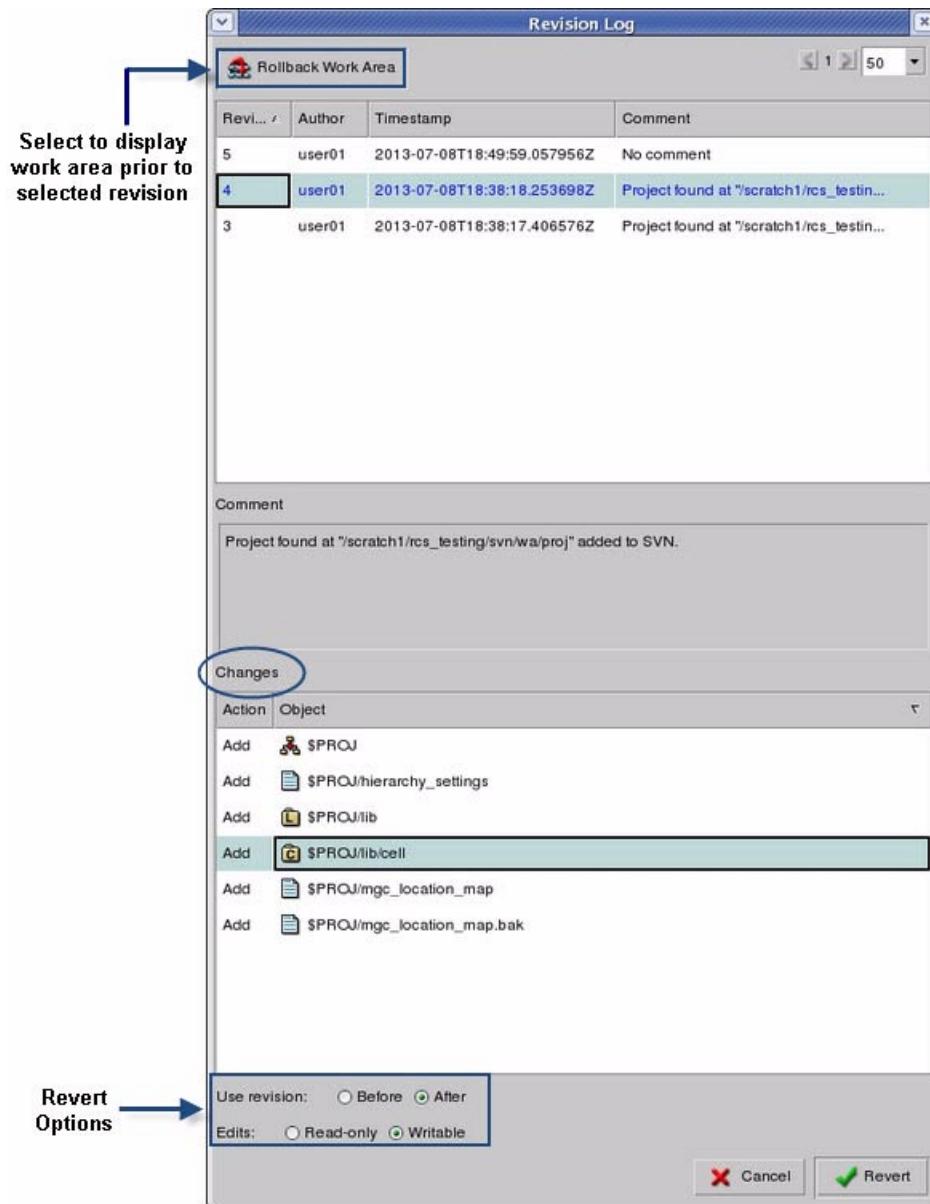
- You must have permissions to update a work area from the revision control system.
- The revision control system must be running at the time of work area creation.

Procedure

1. Select **Revision Control > Revision Log**. The Revision Log Dialog Box displays as shown in [Figure 5-16](#).
2. Choose a revision from the displayed list.
The revision in blue text is the revision of the selected object within the work area.
Selecting a revision completes the dialog box.
3. Select an Object in the **Changes** section, if required.
4. Choose revert options.
5. Select the **Revert** button.

Results

Figure 5-16. Revision Log Dialog Box (SVN)



You have displayed all the revisions for a selected object or you have reverted a selected revision.

Related Topics

[Subversion Revision Control Procedures](#)

Common Revision Control Procedures

There are certain common tasks associated with using ClioSoft or Subversion revision control in Pyxis Project Manager.

Using Automatic Check In	390
Checking Out Objects	391
Canceling an Object Check Out	391
Checking In Objects	393
Updating an Object	395
Managing Hierarchy	395
Data Management	397

Using Automatic Check In

Prompts the user to check in any modified database when exiting the software. Unmodified data is just unlocked in the revision control repository.

Prerequisites

- You have an editable file opened in any Pyxis tool under revision control, which requires that the file is checked out before it can be modified.

Procedure

1. Perform an action to modify the revision controlled file or files.
2. Exit the application. This action displays the Check In Edits dialog box as shown in [Figure 5-18](#), which provides a list of the edits about to be committed along with a default comment.

Results

- Pressing the **Yes** button saves the edits automatically to the revision control system.
- Pressing the **No** button prevents the edits from being checked in and the files remain locked in the revision control system.

Related Topics

[Checking Out Objects](#)[Canceling an Object Check Out](#)[Checking In Objects](#)[Updating an Object](#)[ClioSoft Revision Control Procedures](#)[Subversion Revision Control Procedures](#)

Checking Out Objects

Check out a flat or hierarchical object using the Revision Control menu.

Alternatively, you can open a schematic or layout object for editing to perform check out.

Prerequisites

- You must have permissions to checkout objects from the revision control system.
- The revision control system must be running at the time of checkout.
- The object(s) must be managed and unlocked.

Procedure

1. Select the object(s) to be checked out.
2. Select **Revision Control > Flat > Check Out** or **Revision Control > Hierarchical > Check Out**.

Results

The objects are checked out and ready for editing.

Related Topics

[Revision Control Menu in Pyxis Project Manager](#)

[Checking Out Objects](#)

[Using Automatic Check In](#)

Canceling an Object Check Out

Discard all edits and release the object for edits by other users.

Cancel check out prompts you with a list of edits to be discarded. You can choose between flat or hierarchical object cancel check out. This operation can only be performed via the Revision Control menu.

Caution



Saving an object does not automatically relinquish the lock.

Selecting **Cancel Check Out** on an object which is Modified (and Locked by Me) also reverts edits for that object.

An administrative cancel check out is available to unlock objects within the repository. For example, if the lock owner is unavailable to unlock the objects, then the administrator is able to

do so. Access requires administrative privileges in both Pyxis and Subversion or ClioSoft. To enter admin mode in Pyxis, select **Setup > Admin Login** and enter your password.

Prerequisites

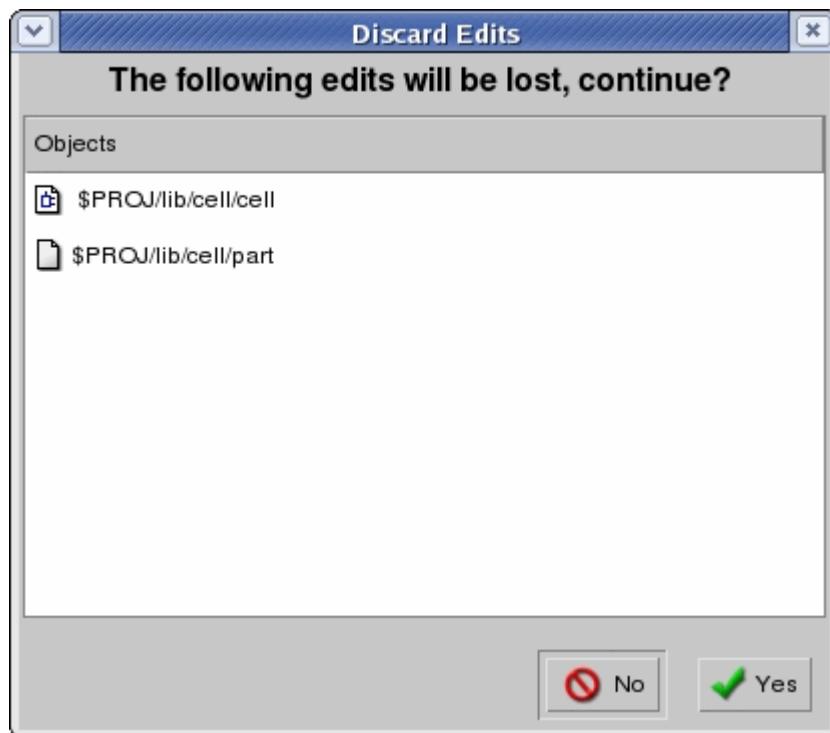
- You must have permission to perform operations on the revision control system.
- The revision control system must be running at the time of cancellation.

Procedure

1. Select the object(s) for which to cancel check out.
2. Select **Revision Control > Flat > Cancel Check Out** or **Revision Control > Hierarchical > Cancel Check Out**
3. Verify that the objects you want to cancel are displayed in the Discard Edits prompt as shown in [Figure 5-17](#).
4. Select **Yes** to cancel the check out and discard the edits.

Results

Figure 5-17. Discard Edits Prompt



The check out operation is canceled.

Related Topics

[\\$cancel_checkout_objects_for_rc\(\)](#)

[Revision Control Menu in Pyxis Project Manager](#)

[Checking Out Objects](#)

[Using Automatic Check In](#)

Checking In Objects

Check in modified objects.

You can choose to check in flat or hierarchical objects. During the check in process, any objects that are locked by you become unlocked.

Prerequisites

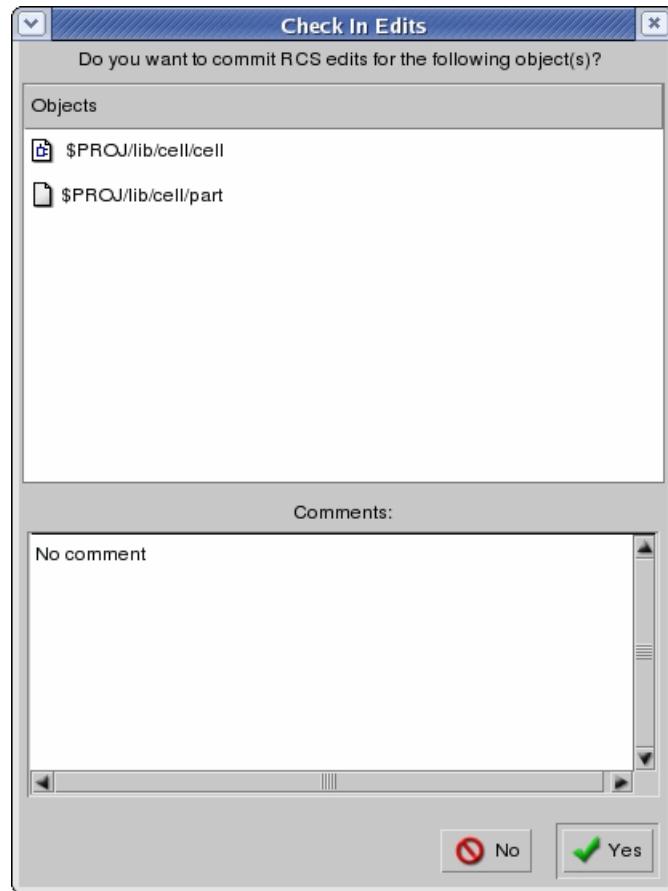
- You must have permissions to check in objects to the revision control system.
- The revision control system must be running at the time of check in.

Procedure

1. Select the object(s) to be checked in.
2. Select **Revision Control > Flat > Check In** or **Revision Control > Hierarchical > Check In**
The Check In Edits prompt displays as shown in [Figure 5-18](#).
3. Verify that the object(s) you want to check in are displayed in the prompt.
4. Optional: Enter a comment for the object(s) that you want to check in.
5. Select **Yes**.

Results

Figure 5-18. Check in Edits Dialog Box



The modified edits are checked in successfully.

Related Topics

[\\$checkin_objects_to_rc\(\)](#)

[Revision Control Menu in Pyxis Project Manager](#)

[Checking Out Objects](#)

[Using Automatic Check In](#)

Updating an Object

Sync changes made in the repository with the work area.

An object that is out-of-date requires updating. You can update a single object using the Flat menu items or update an object and all its contents using the Hierarchical menu items.

Prerequisites

- You must have permissions to update objects from the revision control system.
- The revision control system must be running at the time of update.

Procedure

1. Select the object(s) to be updated.
2. Select **Revision Control > Flat > Update** or **Revision Control > Hierarchical > Update**.

Results

Repository changes are now updated within the current work area.

Related Topics

[\\$update_objects_from_rc\(\)](#)

[Revision Control Menu in Pyxis Project Manager](#)

[Checking Out Objects](#)

[Using Automatic Check In](#)

Managing Hierarchy

Allows you to perform revision control operations based upon the design hierarchy.

Restrictions and Limitations

The tag, branch, and snapshot operations on the design hierarchy are applied to the cell as a whole.

Prerequisites

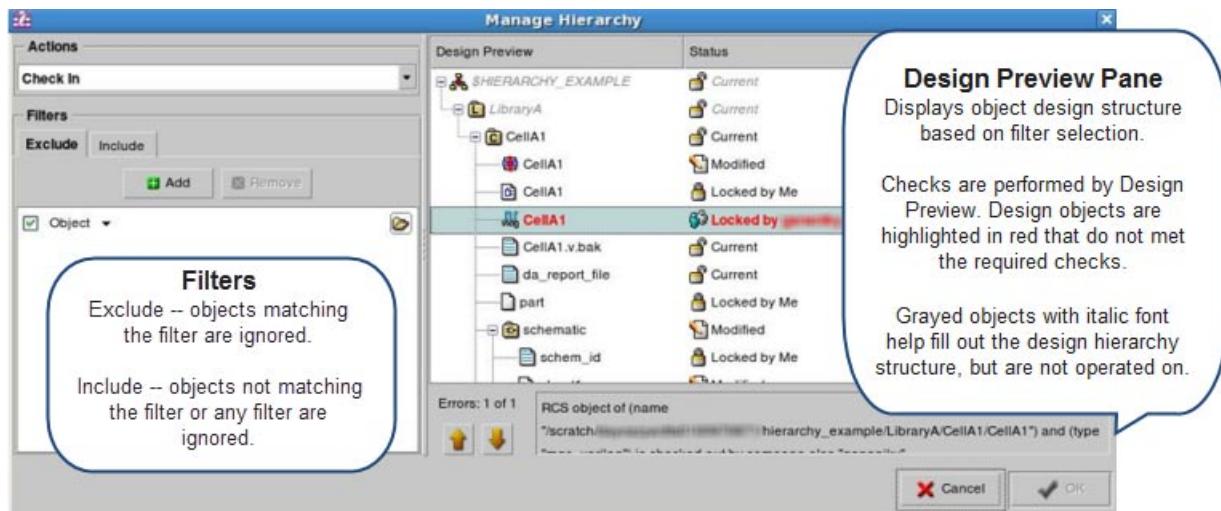
- You must set the appropriate revision control environment variables.
- A root HDO needs to be added to the repository before any revision control operations are performed. If the Manage Hierarchy dialog box is opened for the root hierarchy that is not in the repository, then the root HDO is highlighted with red and an error is reported.

Procedure

1. Select the **Revision Control > Manage Hierarchy...** menu option.
2. Select the appropriate action to perform from the **Actions** drop-down menu. Selecting an operation performs the check and reports errors as shown in [Figure 5-19](#).

The actions available in the drop-down list depend on the revision control selection (ClioSoft or Subversion). Applying a tag, branch, or snapshot action items are only available for ClioSoft.

Figure 5-19. Manage Hierarchy Dialog Box



3. Select the **OK** button to perform the selected operation.

Results

Error messages display in the Errors portion of the dialog box and the **OK** button is not available when errors are detected. The error conditions are displayed at the view level and are not reported in the transcript area of Project Manager.

Scroll through the error messages using the **Next** and **Previous** buttons. Selecting a red item in the Design Preview pane updates the message area with the appropriate message.

The errors are displayed within the current session of the Manage Hierarchy dialog box. When you exit the dialog box and open it again, the errors are cleared and the navigation buttons are disabled.

If no errors are detected, the selected operation is executed.

Related Topics

Revision Control Environment Variables and Setup	Viewing Object Status
Applying Snapshots	Tagging Objects
Setting a Branch	Checking Out Objects
Canceling an Object Check Out	Checking In Objects
Updating an Object	

Data Management

There are several data management functions that you can perform while RCS is enabled including deleting, renaming and moving, and copying.

Deleting a managed object while RCS is enabled deletes it from your work area. It also deletes the object from the repository although your previous versions are accessible to an administrator.

Renaming or moving a managed object while RCS is enabled copies the object to the destination and delete it from the source. This operation changes the repository.

Copying a managed object while RCS is enabled is similar to copying without RCS enabled except that locking the source before the copy is mandatory. This lock ensures that ClioSoft's Smart Cache works properly. At the end of the copy operation, all of the source data remains unchanged.

Related Topics

ClioSoft Revision Control Procedures	Revision Control Menu in Pyxis Project Manager
Common Revision Control Procedures	Subversion Revision Control Procedures

Subversion Administrator Information

There are some administrative revision control tasks for Subversion only.

Subversion Client	398
Creating a Repository	398
Apache Versus SVNServe	399
Qualification Scripts and Revision Control Integration	403

Subversion Client

A common installation for the Subversion client can be found at:

```
$MGC_HOME/bin/svn  
$MGC_HOME/bin/svnadmin  
$MGC_HOME/bin/svnserve  
$MGC_HOME/lib/libsvn*.so
```

The Subversion API that integrates with Mentor Graphics Pyxis tools is version 1.4.

It may be necessary to add `$MGC_HOME/bin` and `$MGC_HOME/lib` to your PATH or `LD_LIBRARY_PATH`.

Related Topics

[Setting Up the Subversion Environment](#)

Creating a Repository

Creating a repository requires the `svnadmin` tool, which comes with the Subversion client. Using URLs is not allowed.

Prerequisites

None.

Procedure

1. If you have Apache set-up, then you may be able to browse the Subversion server and all repositories by pointing your internet browser at your server (e.g., `http://myserver/svn/`).

```
svnadmin create /net/myserver/usr/local/svn/my_repo
```

2. Set permissions for your repository.
3. For some application domains, such as Pyxis Project Manager revision control, it is useful to have subdirectories devoted to a particular project (`mgc_project`). To accommodate this use model, add directories under the repository root.

```
svn mkdir http://myserver/svn/my_repo/main_projects
```

4. In the Pyxis Project Manager revision control use cases, once you have added the subdirectories to the repository being served by the *myserver* Apache process, then you set the Pyxis Project Manager variables to:

```
MGC SVN REPOSITORY=http://myserver/svn/my_repo/main_projects  
AMPLE_PATH=$MGC_HOME/shared/examples/rcs/subversion
```

You can delete the repository using the rm -rf command.

```
rm -rf /net/myserver/usr/local/svn/my_repo
```

Results

A Subversion repository is created.

Related Topics

[Subversion Administrator Information](#)

Apache Versus SVNService

Mentor Graphics recommends using Apache because it appears to be more responsive to concurrent requests than SVNService.

Installing Apache	399
Installing SVNService	401

Installing Apache

The Subversion's Apache module is written specifically for the Apache 2 API so be sure to download Apache 2.X.

Prerequisites

None.

Procedure

1. Download source tar ball from <http://apache.org>.
2. Untar to a directory of your choosing.

```
gtar -zvxf <filename>
```

3. Configure the build.

```
./configure --prefix=/usr/local/apache --enable-modules=all
```

4. Compile.

```
make
```

5. Install.

```
make install
```

6. Start Apache (a warning message about a fully qualified domain name is normal).

```
/usr/local/apache/bin/apachectl start
```

7. Test Apache by going to “//localhost” in your browser.

8. Create a user for Apache (these might already exist).

```
groupadd apache
useradd -g apache -d /usr/local/apache
```

9. Edit the Apache configuration file.

```
vi /usr/local/apache/conf/httpd.conf
```

10. Find “User daemon” and change it to “User apache”.

11. Find “Group daemon” and change it to “Group apache”.

12. Place the following text at the bottom of the httpd.conf file:

```
<Location /subversion>
  DAV svn
  SVNPath /usr/local/subversion/repository/
  AuthType Basic
  AuthName "Subversion repository"
  AuthUserFile /usr/local/apache/auth-file
  Require valid-user
</Location>
```

13. Make a Subversion repository directory.

14. Create the username/password authentication file.

```
/usr/local/apache/bin/htpasswd -c /user/local/apache/auth-file
my_first_user
```

15. Test the installation by navigating to //localhost/subversion.

- You should be prompted for a user name and password (the same ones that you created a few steps ago).
- The page should display with bold text “Revision 0: /”

Results

The Apache server is installed.

Related Topics

[Subversion Administrator Information](#)

[Creating a Repository](#)

Installing SVNServe

An alternative to a full Apache server installation (with the Subversion module) is the program *svnserve*.

Prerequisites

After creating the repository from the previous section, you need to do some additional setup which differs from an Apache configuration.

Use the same paths from [page 398](#):

```
cd /net/myserver/usr/local/svn/my_repo/conf
```

Procedure

1. Edit the file *passwd*.
 - a. Go to the section labelled *[users]*.
 - b. Add a line of the form *name = password* for each username and password to be supported by *svnserve*.

For example:

```
### This file is an example password file for svnserve.  
### Its format is similar to that of svnserve.conf. As shown in  
### the example below it contains one section labelled [users].  
### The name and password for each user follow, one account per  
### line.  
  
[users]  
bob pass1234
```

2. Edit the file *svnserve.conf*.
 - a. Uncomment *anon-access* line and change access to *none*.
 - b. Uncomment *auth-access* line and set to *write*.
 - c. Uncomment *password-db* line and set to *passwd*.
 - d. Uncomment *realm* line and set to a unique string of your choice.

For example, abbreviated:

```
[general]
### These options control access to the repository for
### unauthenticated and authenticated users. Valid values are
### "write", "read", and "none". The sample settings below are
### the defaults.
anon-access = none
auth-access = write
### The password-db option controls the location of the password
### database file. Unless you specify a path starting with a '/',
### the file's location is relative to the conf directory.
### Uncomment the line below to use the default password file.
password-db = passwd
### This option specifies the authentication realm of the
### repository. If two repositories have the same authentication
### realm, they should have the same password database, and
### vice versa. The default realm is repository's uuid.
realm = This is my repository
```

3. Run. Once the configuration information has been updated, running *svnservice* is straight forward. On the machine that serves as the repository:

```
svnservice -d -r /net/myserver/usr/local/svn/my_repo
```

Depending on your network configuration, the machine that is running *svnservice* is part of the URLs used to access the database:

```
svn ls svn://myserver
```

If *svnservice* is invoked without the *-r* option, then all file system paths on the host machine are accessible. The user would simply supply the full path to the repository.

```
svn ls svn://myserver/usr/local/svn/my_repo
```

Specifying a root with *-r* restricts access to directories under that root. If multiple repositories exist under that root, *svnservice* can serve each of them (by specifying the subdir path to that repository). If the *-r* path is to a single repository, then only that one is accessible.

Results

The server is installed.

Related Topics

[Subversion Administrator Information](#)

[Creating a Repository](#)

Integrating Custom Type Data with Revision Control

Integrate user data types.

Prerequisites

None.

Procedure

There are two paths to integrate user data types with PMPN:

1. **Registrar-based Custom Type Creation.** This procedure is discussed in the *Registrar User's and Reference Manual* available via the PDF Bookcase or SupportNet. See the tutorial in Chapter 2 for details, including setting the custom toolbox path and the **\$MGC_TYPE_REGISTRY** environment variable.

- OR -
2. **PMPN Custom Type Creation.** The new system involves fewer steps and centralizes custom types:
 - a. `setenv MGC_CUSTOM_TYPE_DIR /dir/for/custom/types.`
 - b. Invoke *dmgr_ic*.
 - c. Select **Setup > Admin Login** and fill out your site administrative password.
 - d. Select **Setup > New Type** or **Setup > Edit Type** and follow the prompts.

Results

User data types are now integrated with PMPN.

Related Topics

[Revision Control](#)

Qualification Scripts and Revision Control Integration

You are no longer required to implement your own checkout utility functions.

Example 5-3. Registrar-based Qual Script

Given the new function `$rcs_qual_checkout()`, the Registrar-based qual script example would be modified to look like this:

```
/* Qualification script for the tool reg_tut.
This script determines from where the tool was invoked
and then calls the executable script, with an argument
if appropriate. */

switch($get_object_type())
{
    case "reg_tut_data":
        $message("Navigator invocation");
        $rcs_qual_checkout($get_object_pathname(), $get_object_type());
        $invoke_tool("/scratch1/training/reg_tut/work_to_play",
                    $get_object_pathname());
        break;

    case "reg_tut_tool":
        $message("Tool window invocation");
        $invoke_tool("/scratch1/training/reg_tut/work_to_play",
                    $get_object_pathname());
        break;

    case VOID:           /* an error occurred */
        $message ("You must select an object to invoke.");
        break;

    default:           /* default case */
        $message($strcat("Cannot invoke tool on ",
                        $get_object_pathname()));
}
```

Example 5-4. PMPN-based Qual Script

The PMPN-based qual script example would be modified to look like this:

```
extern fileset_members;

switch($get_object_type())
{
    case "custom_data":
        fileset_members =
            $$get_fileset_members($get_object_pathname(), "custom_data");

        if (length(fileset_members) > 0 {
            $message("Navigator invocation");
            $rcs_qual_checkout($get_object_pathname(), "custom_data");
            $invoke_tool("/user/bin/emacs", fileset_numbers[0]);
        }
        else { /* this shouldn't be possible */
            $message(
                $strcat("Cannot invoke tool -- failed to find file for object",
                       $get_object_pathname()));
        }
        break;

    /* RESERVED_FOR_FUTURE_OBJECT_TYPES */

    case "emacs":
        $message("Tool window invocation.");
        $invoke_tool("/user/bin/emacs");
        break;

    case VOID: /* an error occurred */
        $message("You must select an object to invoke.");
        break;

    default: /* default case */
        $message($strcat("Cannot invoke tool on ",
                       $get_object_pathname()));
}
}
```

Related Topics

[Revision Control](#)

ClioSoft Troubleshooting

The following table provides solutions to common ClioSoft problems.

Table 5-4. Troubleshooting

Problem	Solution
Modified object shows Current when not checked out.	Either check out the object or set the environment variables SPX_CI_MODIFIED and SOS_CHECK_READ_ONLY to display all such objects as modified.

Table 5-4. Troubleshooting

Problem	Solution
Admin cancel check out is enabled in the revision control menu but it does not work.	Admin operations within SOS require admin privileges. Consult the SOS documentation for more information.
On some occasions, the object status within the Project Manager Navigator does not update after operations.	Doing a refresh (Windows > Refresh All or F5) updates the status for all objects.

Related Topics

[Revision Control](#)

Chapter 6

OpenAccess Import/Export

The Pyxis Custom Design flow supports the ability to convert to and from OpenAccess databases.

In v10.5_1, this ability is limited to databases produced in Pyxis Layout (mask data) and Pyxis Schematic (design data) only.

OpenAccess Import Prerequisites	407
Using OpenAccess Schematic Import	407
Using OpenAccess Layout Import	410
OpenAccess Export Prerequisites	411
Using OpenAccess Schematic Export	411
Using OpenAccess Layout Export	413
OpenAccess Via Export/Import	414

OpenAccess Import Prerequisites

To import OpenAccess data, you need to do the following:

- Create a Pyxis Project Manager Project Navigator project.
- Ensure that the OpenAccess database you are importing has a correct *lib.defs* file available.
- Verify that the project associates with a Mentor converted PDK and has an appropriate configuration.
- Layout only: OpenAccess import does not work on technology libraries. It only works with design libraries with techlibs pre-existing.

Related Topics

[Using OpenAccess Schematic Import](#)

[Using OpenAccess Layout Import](#)

Using OpenAccess Schematic Import

Use OpenAccess Schematic Import to import a library of schematics. The schematic importer converts all symbols and their parameters prior to converting the schematics.

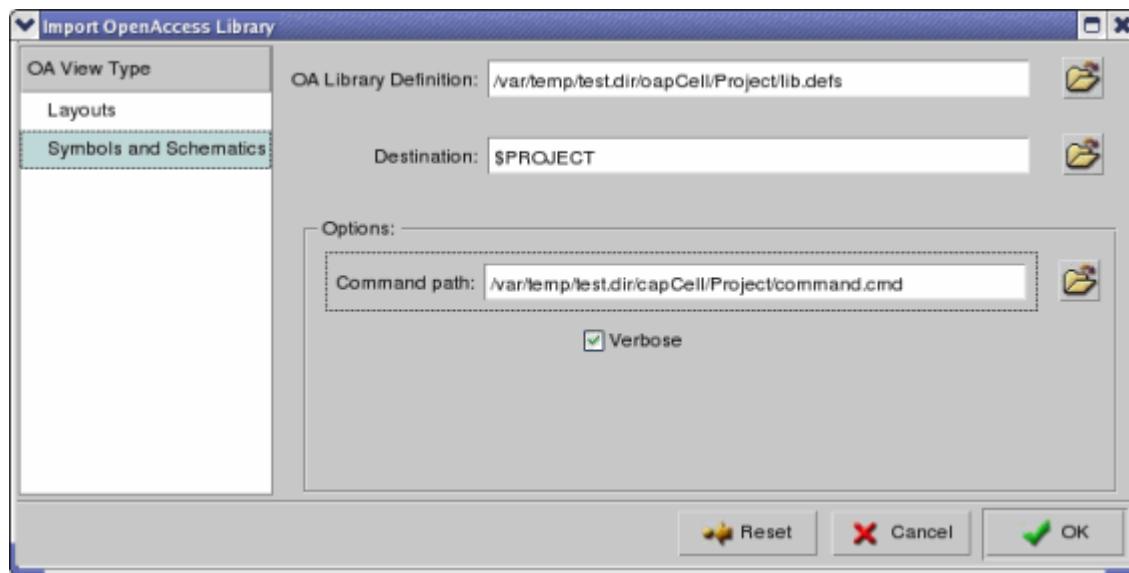
Prerequisites

- Before beginning to import schematics and symbols, be sure you have performed all prerequisites as outlined in the section, “[OpenAccess Import Prerequisites](#)” on page 407.

Procedure

1. Select **File > Import > OpenAccess** or **Tools > Import OA** (for symbols and schematics only).
2. Choose **Symbols and Schematics** from the OA View Type column on the left, as shown in [Figure 6-1](#).

Figure 6-1. Import OpenAccess Library



3. Select an **OA Library Definition** (*lib.defs*) file associated with the OpenAccess database. This field is a required selection.
4. Choose the **Destination** path to the specified project. This field is a required selection.
5. Designate the **Command path**. This file can contain directives. This field is an optional selection. This path is independent from the command path for OpenAccess Layouts.

Consider the directive “`deleteProperty ignore`”. This directive asks the importer to delete any property called “`ignore`” during the importing process.

Note

 EDIF Importer commands are applicable as directives in the optional command file.

You can find a copy of the log file under **Tools > Log Files**.

Related Topics

[Running the OA Schematic Importer](#)

[*Pyxis Schematic User's Manual*](#)

[EDIF Converter Quick Start Guide](#)

Using OpenAccess Layout Import

The import process supports importing a mask data library into a Pyxis Project Manager library. All OpenAccess objects that are equivalent or that can be mapped to Pyxis data objects are converted. Any object that cannot be converted is skipped.

The primary backend import types are maskable layout objects, instances, arrays, kit-compatible devices, the most common via types as well as properties associated with such objects, nets, and groups where they map to Pyxis groups.

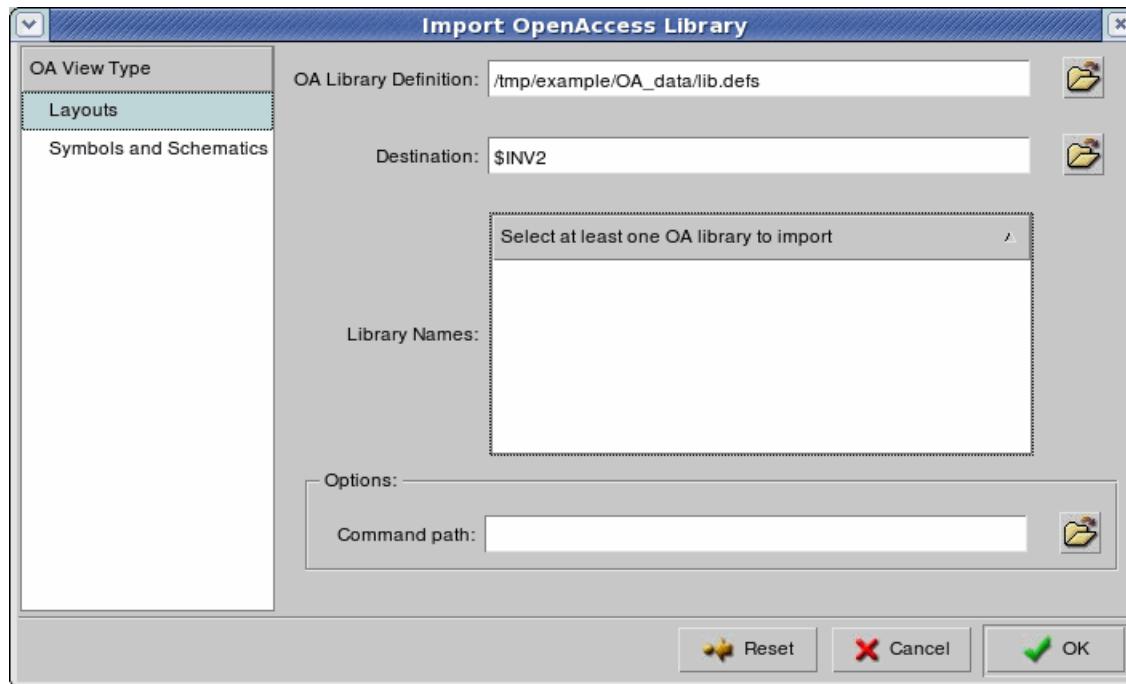
Prerequisites

- Before beginning to import mask data, be sure you have performed all prerequisites as outlined in the section, “[OpenAccess Import Prerequisites](#)” on page 407.

Procedure

1. Select **File > Import > OpenAccess**.
2. Choose **Layouts** from the OA View Type column on the left, as shown in [Figure 6-2](#) below.

Figure 6-2. Import OpenAccess Library Layouts Dialog Box



3. Select an **OA Library Definition** (*lib.defs*) file associated with the OpenAccess database. This field is a required argument.

4. Choose the **Destination** path to the root of a Project Navigator project. The converted library is placed at this location.
5. Select one or more OpenAccess libraries to import from the **Library Names** field. This field is populated with all libraries found in the *lib.defs* file when you choose it in the **OA Library Definition** field.
6. Designate the **Command path**. This file can contain directives. This field is an optional argument. This path is independent from the command path for OpenAccess Schematics.

Note



The directives that modify the OA Layout to Pyxis database converter are AMPLE commands.

7. Press **OK**.

You can find a copy of the log file under **Tools > Log Files**.

Related Topics

[\\$set_oareader_map\(\)](#).

OpenAccess Export Prerequisites

To export OpenAccess data, you need to do the following:

- Ensure that the OpenAccess data destination is writable.
- Verify that the project associates with a Mentor converted PDK and has an appropriate configuration.
- Schematic Only: Ensure that the OpenAccess data is parallel to Mentor component data inside the Mentor converted PDK.
- Layout Only: Find the location of an OpenAccess library whose technology database corresponds to the Mentor PDK.

Related Topics

[Using OpenAccess Schematic Export](#)

[Using OpenAccess Layout Export](#)

Using OpenAccess Schematic Export

Use the OpenAccess schematic export process to export a library of schematics.

Prerequisites

- Before beginning to export schematics and symbols, be sure you have performed all prerequisites as outlined in the section “[OpenAccess Export Prerequisites](#)” on page 411.

Procedure

1. Select **File > Export > OpenAccess** or **Tools > Export OA** (for symbols and schematics only).
2. Choose **Symbols and Schematics** from the OA View Type column on the left, as shown in [Figure 6-3](#) below.

Figure 6-3. Export OpenAccess Symbols and Schematics Dialog Box



3. Specify a **Destination Path**. This is the destination path to where the library is exported.
4. Specify the Project Manager project from which to export in the **Source project** field.
5. Enable the **Overwrite** option to overwrite the old exported data in the same destination path.
6. Click **OK** to run the OA exporter.

Results

Once run, the exporter transcribes the time it started, and when finished, it also transcribes a success or error message. All of this information is sent to a log file. The log file for the exporter is under **Tools > Log Files**. In the event of an error in the export process, the log file is opened and scrolled to the summary at the end.

Related Topics

[Running the OA Schematic Exporter](#)

Using OpenAccess Layout Export

Use OpenAccess layout export for backend data.

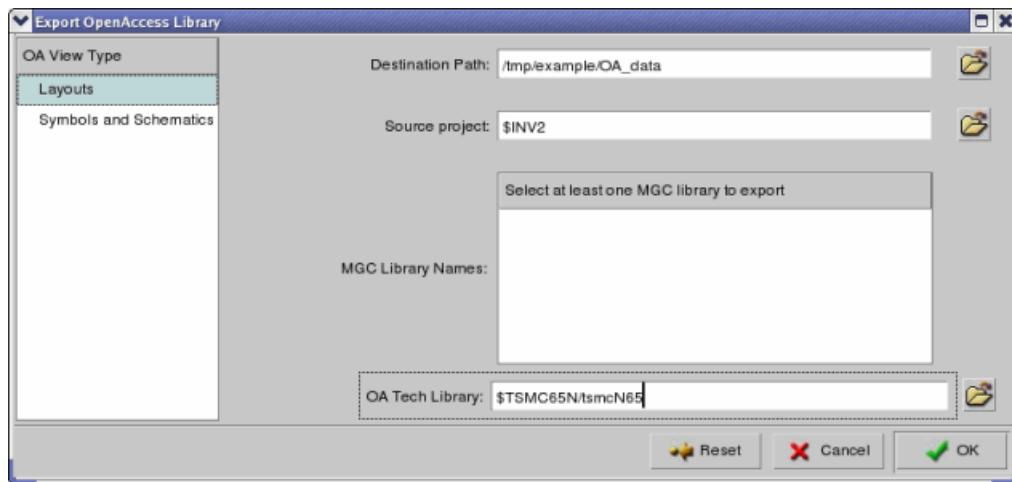
Prerequisites

- Before beginning to export mask data, be sure you have performed all prerequisites as outlined in the section “[OpenAccess Export Prerequisites](#)” on page 411.

Procedure

- Select **File > Export > OpenAccess**.
- Choose Layouts from the OA View Type column on the left, as shown in [Figure 6-4](#).

Figure 6-4. Export OpenAccess Layouts Dialog Box



- Select a **Destination Path**. This is the path to the directory where all libraries are exported. The generated *lib.defs* file is placed here as well.
- Select the **Source project** from which to export selected libraries.
- Select the MGC libraries you wish to export from the **MGC Library Names** field. This field is populated with all libraries found in the source project when you choose a project.
- Choose an **OA Tech Library**. This is the path to the OA library whose technology database is related to the Mentor PDK for the targeted project.

Related Topics

[OpenAccess Export Prerequisites](#)

[Pyxis Layout User's Manual](#)

OpenAccess Via Export/Import

Only Mentor design kits are supported for exporting and importing OpenAccess vias to ensure that there are matched via definitions for OA technology and the Pyxis Custom Design process.

To guarantee the via internals are preserved after exporting and importing, Mentor supports the matching of Pyxis Custom Design via devices, objects, and instances with oaCustomVia in the OA database.

OA Via Export	414
OA Via Import	414

OA Via Export

There are two OA via export cases:

1. Export \$via device to oaCustomVia

Supported via devices are sorted based on type and size. The device internals to the OA cells are written and exported to the output OA library. The oaCustomViaDef is appended to the technology file and then the \$via device is exported as a special oaCustomVia to the design. Special properties are attached to the oaCustomVia that are used to convert the oaCustomVia back to \$via device when importing.

2. Export via object or instance to oaCustomVia

If the via object/instance's name matches the name of an oaCustomViaDef in the OA technology file, the via object/instance is translated as oaCustomVia with the matched oaCustomViaDef.

If there is no matched oaCustomViaDef with such a name, the OA design with the name is created in the output OA library and the via cell internals are written to it. The newly created oaCustomViaDef with the name is appended to the OA technology file and the via object/instance are translated as oaCustomVia with the new oaCustomViaDef.

Related Topics

[OA Via Import](#)

OA Via Import

There are two OA via import cases:

1. Import oaStdVia to \$via device

If the OA design is written by the Pyxis OA export, then there should not be any oaStdVia in the OA design. Mentor supports reading oaStdVia even if the OA design is not produced by Pyxis by matching oaStdViaDef name in the OA technology to the \$via device type name in the Pyxis process.

There is no guarantee that the device internals are the same, and you receive this warning message in the transcript.

```
//OA Cellview . . . appears to have been generated by an external  
tool. Making best effort import.
```

2. Import oaCustomVia to via object/instance

If the oaCustomVia has special properties (an original \$via device converted as an oaCustomVia by OA export), then import it by reading the properties and reconverting it to the \$via device.

Otherwise, match its oaCustomViaDef name to the via cell name in the Pyxis process. If the via cell name is found, check if the via is suitable to be imported as a via object. If it is suitable, then import it as a via object, else import it as a via instance. If there is no matched via cell found in the process, oaCustomVia is ignored with an error message.

Related Topics

[OA Via Export](#)

Chapter 7

Function Summary

There are many functions that you can use during a Pyxis Project Manager session.

Summary of All Functions	417
Interactive Function Summary	433
Design Object Toolkit Function Summary	440
Configuration Management Toolkit Function Summary	445
Tool Viewpoint Function Summary	448
iDM Interactive Function Summary	449
iDM Toolkit Function Summary	451

Summary of All Functions

The following list describes all functions you can use during a Pyxis Project Manager session

Table 7-1. Pyxis Project Manager Function Summary

Function	Description
\$\$add_configuration_entry()	Adds the specified design object version to the active configuration.
\$add_configuration_entry()	Adds the current version of a design object to the active configuration window.
\$\$add_container()	Creates a basic container design object with an attribute file.
\$add_container()	Creates a basic container design object with an attribute file.
\$\$add_directory()	Creates a container design object without an attribute file.
\$add_directory()	Creates a directory without an attribute file.
\$add_object_property()	Adds a property to the selected design object.
\$\$add_reference()	Creates a reference from the specified source design object to the specified target design object.

Table 7-1. Pyxis Project Manager Function Summary

Function	Description
<code>\$add_reference()</code>	Creates a reference from the selected design object to a target design object.
<code>\$add_reference_property()</code>	Adds a property to the selected reference.
<code>\$add_toolbox()</code>	Appends the specified toolbox to the current toolbox search path.
<code>\$add_versions()</code>	Adds the specified versions of the selected design object to the active configuration window.
<code>\$browse_for_object()</code>	Brings up an object browser dialog.
<code>\$\$build_configuration()</code>	Adds secondary entries to the active configuration, based on the build rules of the primary entries.
<code>\$build_configuration()</code>	Adds secondary entries to the active configuration window, based on the build rules of the primary entries.
<code>\$\$change_configuration_references()</code>	Changes the references of each entry in the active configuration.
<code>\$change_configuration_references()</code>	Changes the references of each entry in the active configuration window.
<code>\$\$change_design_object_references()</code>	Changes the references of the specified design objects.
<code>\$change_design_object_references()</code>	Changes the references of the specified design objects.
<code>\$change_location_map_entry()</code>	Overrides a location map entry in memory.
<code>\$\$change_object_name()</code>	Renames the specified design object.
<code>\$change_object_name()</code>	Renames the specified design object.
<code>\$change_object_property()</code>	Changes a property value of the selected design object.
<code>\$\$change_object_references()</code>	Changes the references of the specified design object.
<code>\$change_object_references()</code>	Changes the references of the selected design object.
<code>\$change_protection()</code>	Changes the protection of the selected design object.
<code>\$change_reference_property()</code>	Changes a property of the selected reference.

Table 7-1. Pyxis Project Manager Function Summary

Function	Description
<code>\$change_reference_state()</code>	Changes the state of the selected reference.
<code>\$change_version_depth()</code>	Changes the version depth of the selected design object.
<code>\$check_references()</code>	Provides a simplified interface to check and fix the broken references of the selected design objects.
<code>\$check_registries()</code>	Checks the \$MGC_HOME/registry directory.
<code>\$\$clear_entry_filter()</code>	Resets all filters of a primary entry's build rules.
<code>\$\$clear_global_status()</code>	Clears the global toolkit status stack.
<code>\$\$clear_monitor()</code>	Clears all text from the configuration's monitor window.
<code>\$\$close_configuration()</code>	Removes the lock on the active configuration.
<code>\$\$close_versioned_object()</code>	Closes a versioned design object that encapsulates data from a non-Mentor Graphics application.
<code>\$close_window()</code>	Closes the active window.
<code>\$\$convert_configuration_references()</code>	Converts the references of each entry in the active configuration.
<code>\$convert_configuration_references()</code>	Automatically converts reference pathnames on all objects in the configuration from hard reference pathnames to soft reference pathnames using the active location map.
<code>\$\$convert_object_references()</code>	Converts the references of the specified design object(s).
<code>\$convert_object_references()</code>	Automatically converts the reference pathnames of selected design objects from hard to soft using the current location map.
<code>\$\$copy_configuration()</code>	Copies all design object versions in the active configuration, to a target location.
<code>\$copy_configuration()</code>	Copies all design object versions in the active configuration window to a target location.
<code>\$\$copy_design_object()</code>	Copies one or more design objects to a container.
<code>\$copy_design_object()</code>	Copies one or more design objects to a container.

Table 7-1. Pyxis Project Manager Function Summary

Function	Description
<code>\$\$copy_object()</code>	Copies the specified design object to a target location.
<code>\$copy_object()</code>	Copies the selected design objects to a target location.
<code>\$copy_version()</code>	Copies a single selected version of the design object to a target location.
<code>\$\$create_configuration()</code>	Creates a configuration object.
<code>\$\$create_versioned_object()</code>	Creates a versioned design object that encapsulates data from a non-Mentor Graphics application.
<code>\$\$delete_configuration()</code>	Deletes all design object versions in the active configuration from the disk.
<code>\$delete_configuration()</code>	Deletes all design object versions in the active configuration window from the disk.
<code>\$delete_design_object()</code>	Deletes the specified design objects.
<code>\$delete_excess_versions()</code>	Deletes all but a specified number of versions of the selected design object(s).
<code>\$\$delete_object()</code>	Deletes the specified design object.
<code>\$delete_object()</code>	Deletes the selected design object.
<code>\$\$delete_object_property()</code>	Deletes the specified property from the specified design object.
<code>\$delete_object_property()</code>	Deletes the specified property from the selected design object.
<code>\$\$delete_reference()</code>	Deletes the specified reference from the specified source design object.
<code>\$delete_reference()</code>	Deletes the selected reference from the source design object.
<code>\$\$delete_reference_handle()</code>	Deletes a reference from the specified design object by using the reference target handle.
<code>\$\$delete_reference_property()</code>	Deletes a property from the specified reference.
<code>\$delete_reference_property()</code>	Deletes the specified property from the selected reference.
<code>\$\$delete_reference_property_handle()</code>	Deletes a property from the specified reference by using the reference target handle.

Table 7-1. Pyxis Project Manager Function Summary

Function	Description
<code>\$\$delete_version()</code>	Deletes the specified version of the specified design object.
<code>\$delete_version()</code>	Deletes the selected design object version.
<code>\$\$delete_version_property()</code>	Deletes a property from the specified version.
<code>\$\$duplicate_object()</code>	Duplicates a design object.
<code>\$edit_file()</code>	Opens a file in edit or read-only mode.
<code>\$empty_trash()</code>	Deletes the contents of the trash window.
<code>\$explore_contents()</code>	Navigates down one level of the containment hierarchy.
<code>\$explore_parent()</code>	Navigates up one level of the containment or reference network.
<code>\$explore_reference_parent()</code>	Navigates to and displays the contents of the parent directory of the referenced object.
<code>\$explore_references()</code>	Navigates down one level of the reference network.
<code>\$export_location_map()</code>	Causes the current location map to be written to a temp file.
<code>\$find_references()</code>	Finds the references for the selected part
<code>\$\$fix_relative_path()</code>	Generates an absolute pathname from a relative pathname.
<code>\$\$freeze_configuration()</code>	Freezes all design object versions in the active configuration.
<code>\$freeze_configuration()</code>	Freezes all design object versions in the active configuration window and creates a new version of the configuration.
<code>\$\$freeze_version()</code>	Freezes a version of the specified design object.
<code>\$freeze_version()</code>	Freezes the selected design object version.
<code>\$get_area_selected_objects()</code>	Returns the pathnames and types of the selected design objects in the navigator window.
<code>\$\$get_children()</code>	Returns the children of a configuration entry.
<code>\$\$get_configuration_entries()</code>	Returns all entries in the active configuration.
<code>\$\$get_configuration_path()</code>	Returns the soft pathname of the active configuration.

Table 7-1. Pyxis Project Manager Function Summary

Function	Description
<code>\$\$get_container_contents()</code>	Returns the name, type, and version of the objects in the specified container.
<code>\$\$get_date_last_modified()</code>	Returns the date that the specified design object was last saved to disk.
<code>\$get_default_tool()</code>	Returns the default tool for the specified type.
<code>\$\$get_entry_version()</code>	Returns the version number of the target object of a current entry in the active configuration.
<code>\$\$get_fileset_members()</code>	Returns the fileset members of the specified design object.
<code>\$\$get_hard_name()</code>	Returns the hard pathname equivalent of any pathname.
<code>\$\$get_location_map()</code>	Returns the absolute path of the current location map, and the values of its entries.
<code>\$\$get_monitor_error_count()</code>	Returns the number of errors processed by the monitor.
<code>\$\$get_monitor_flag()</code>	Returns the Boolean value of the specified configuration monitoring attribute.
<code>\$\$get_monitor_verbosity()</code>	Returns the value of the monitor verbosity.
<code>\$\$get_monitor_warning_count()</code>	Returns the number of warnings processed by the monitor.
<code>\$get_navigator_directory()</code>	Returns the soft pathname of the current navigator directory.
<code>\$get_navigator_directory_hard()</code>	Returns the hard pathname of the current navigator directory.
<code>\$\$get_object_current_version()</code>	Returns the current version of the specified design object.
<code>\$\$get_object_parent_path()</code>	Returns the pathname of the parent of the specified design object.
<code>\$\$get_object_path_filter()</code>	Returns the value of the object path filter for the specified primary entry.
<code>\$get_object.pathname()</code>	Returns the absolute pathname of the specified design object during data-centered tool invocation, or the absolute pathname of the active tool viewpoint during tool-centered tool invocation.

Table 7-1. Pyxis Project Manager Function Summary

Function	Description
<code>\$\$get_object_properties()</code>	Returns the properties of the specified design object.
<code>\$\$get_object_property_filter()</code>	Returns the value of the object property filter for the specified primary entry.
<code>\$\$get_object_property_value()</code>	Returns a property value of the specified design object.
<code>\$\$get_object_protection()</code>	Returns the protection status of the specified design object.
<code>\$\$get_object_references()</code>	Returns the references of the specified design object.
<code>\$\$get_object_type()</code>	Returns the design object type of the specified design object.
<code>\$get_object_type()</code>	Returns the type of the specified design object during data-centered tool invocation, or the type of the active tool viewpoint during tool-centered tool invocation.
<code>\$\$get_object_type_filter()</code>	Returns the value of the object type filter for the specified primary entry.
<code>\$get_object_version()</code>	Returns the version number of the specified design object during data-centered tool invocation, or the version number of the active tool viewpoint during tool-centered tool invocation.
<code>\$\$get_object_versions()</code>	Returns the versions of the specified design object.
<code>\$\$get_parent_entry()</code>	Returns the parent of the specified configuration entry.
<code>\$\$get_primaries()</code>	Returns the primary entries in the active configuration.
<code>\$\$get_reference_properties()</code>	Returns the properties of the specified reference.
<code>\$\$get_reference_properties_handle()</code>	Returns the properties of the specified reference by using the target reference handle.
<code>\$\$get_reference_property_filter()</code>	Returns the value of the reference property filter for the specified primary entry.
<code>\$\$get_reference_traversal()</code>	Returns the reference traversal of the specified primary entry.

Table 7-1. Pyxis Project Manager Function Summary

Function	Description
<code>\$\$get_secondaries()</code>	Returns the secondary entries of the specified primary entry.
<code>\$\$get_soft_name()</code>	Returns the soft pathname equivalent of a pathname.
<code>\$\$get_status_code()</code>	Returns the error message code at the top of the status stack.
<code>\$\$get_status_code_stack()</code>	Returns all of the error message codes on the status code stack.
<code>\$\$get_status_messages()</code>	Returns the error messages on the status stack.
<code>\$get_subinvoke_mode()</code>	Returns the current tool invocation method.
<code>\$\$get_target_path()</code>	Returns the target path of the specified primary configuration entry.
<code>\$get_toolbox_search_path()</code>	Returns the current toolbox search path.
<code>\$get_tool_pathname()</code>	Returns the absolute pathname of the active tool viewpoint during data-centered or tool-centered tool invocation.
<code>\$get_tool_script()</code>	Returns the absolute pathname of the active tool viewpoint's qualification or termination script.
<code>\$get_tool_type()</code>	Returns the type of the active tool viewpoint during data-centered or tool-centered tool invocation.
<code>\$\$get_type_properties()</code>	Returns all the properties associated with that type.
<code>\$\$get_type_property_value()</code>	Returns the value of the property type.
<code>\$\$get_version_depth()</code>	Returns the version depth of the specified design object.
<code>\$\$get_version_properties()</code>	Returns the properties of the specified design object version.
<code>\$\$get_working_directory()</code>	Returns the value of the MGC working directory.
<code>\$goto_directory()</code>	Navigates to the specified directory.
<code>\$\$handle_map_error()</code>	Lets the user determine how to proceed when an error occurs in reading a location map.
<code>\$\$has_object_property()</code>	Checks if a property exists on the specified design object.

Table 7-1. Pyxis Project Manager Function Summary

Function	Description
<code>\$\$has_reference_property()</code>	Checks if a property exists on the specified reference.
<code>\$\$has_reference_property_handle()</code>	Checks if a property exists on the specified reference by using the target reference handle.
<code>\$hide_secondary_entries()</code>	Hides the secondary entries in the active configuration window.
<code>\$hide_monitor()</code>	Hides the configuration's monitor window.
<code>\$invoke_bgd_tool()</code>	Invokes an executable file as a background process.
<code>\$invoke_tool()</code>	Opens a new shell window and invokes an executable file.
<code>\$\$is_build_consistent()</code>	Checks if the active configuration is consistent.
<code>\$\$is_build_valid()</code>	Checks if the active configuration is valid.
<code>\$\$is_configuration_edited()</code>	Checks if the active configuration has been changed and needs to be saved.
<code>\$\$is_configuration_frozen()</code>	Checks if the active configuration is frozen.
<code>\$\$is_configuration_locked()</code>	Checks if the active configuration is locked.
<code>\$\$is_container()</code>	Checks if the specified design object is a container
<code>\$\$is_directory()</code>	Checks if the specified design object is a directory.
<code>\$\$is_entry_container()</code>	Checks if the specified configuration entry is a container.
<code>\$\$is_entry_fixed()</code>	Checks if the specified configuration entry is a fixed entry.
<code>\$\$is_entry_primary()</code>	Checks if the specified configuration entry is a primary entry.
<code>\$\$is_entry_retargetable()</code>	Checks if the specified configuration entry is re-targetable.
<code>\$\$is_object_released()</code>	Checks if the specified design object is in a released state.
<code>\$\$is_read_protected()</code>	Checks if the specified design object can be opened for reading.
<code>\$\$is_relative_path()</code>	Indicates if the specified pathname is a relative pathname.

Table 7-1. Pyxis Project Manager Function Summary

Function	Description
\$\$is_type_versioned()	Checks if the type is versioned.
\$\$is_writable()	Checks if the current process can write to the specified design object.
\$\$is_write_protected()	Checks if the specified design object can be opened for writing.
\$list_references()	List the all references for a component.
\$load_registry()	Loads a type registry into the current session's type manager.
\$\$lock_configuration()	Locks all design object versions in the active configuration.
\$lock_configuration()	Locks all design object versions in the active configuration window.
\$\$lock_object()	Locks the specified design object.
\$maintain_hierarchy()	Maintains a parts directory hierarchy upon release.
\$\$monitor_global_status()	Reports any errors currently on the global toolkit status stack.
\$\$move_design_object()	Moves one or more design objects to a new container.
\$move_design_object()	Moves one or more design objects to a new container.
\$\$move_object()	Moves the specified design objects to a target location.
\$move_object()	Moves the selected design object to a target location.
\$\$object_complete()	Checks if the specified design object is complete.
\$\$object_exists()	Checks if the specified design object exists.
\$\$open_configuration()	Opens the specified configuration object.
\$open_configuration_window()	Opens a new or existing configuration object.
\$open_navigator()	Opens a navigator.
\$open_object()	Opens the selected design object.
\$open_read_only_editor()	Invokes the read-only ASCII editor on the selected object.

Table 7-1. Pyxis Project Manager Function Summary

Function	Description
<code>\$open_session_monitor()</code>	Makes the session monitor window visible.
<code>\$\$open_tool()</code>	Invokes the specified tool.
<code>\$open_tool()</code>	Invokes the selected tool.
<code>\$open_tools_window()</code>	Opens a tools window.
<code>\$open_trash_window()</code>	Opens a trash window.
<code>\$open_types_window()</code>	Displays the types currently loaded in the Pyxis Project Manager.
<code>\$\$open_versioned_object()</code>	Opens a versioned design object that encapsulates data from a non-Mentor Graphics application.
<code>\$\$prune_design_hierarchy()</code>	Prunes off back versions of a design object.
<code>\$\$read_map()</code>	Reads the ASCII location map file into memory.
<code>\$read_map()</code>	Reads the ASCII location map file into memory.
<code>\$\$release_configuration()</code>	Releases all design object versions in the active configuration to a target location.
<code>\$release_configuration()</code>	Releases the design object versions in the active configuration window to a target location.
<code>\$\$release_object()</code>	Releases the current version of the specified objects to the specified destination directory.
<code>\$release_object()</code>	Releases the current version of the selected objects to the specified destination directory.
<code>\$\$remove_configuration_entry()</code>	Removes the specified entry from the active configuration.
<code>\$remove_configuration_entry()</code>	Removes an entry from the active configuration window.
<code>\$remove_toolbox()</code>	Removes a toolbox from the toolbox search path.
<code>\$report_configuration_info()</code>	Displays the status of the active configuration window.
<code>\$\$report_configuration_references()</code>	Reports the references of entries in the active configuration.

Table 7-1. Pyxis Project Manager Function Summary

Function	Description
<code>\$report_configuration_references()</code>	Reports the references of entries in the active configuration window.
<code>\$report_entry_info()</code>	Displays the status of the selected entries in the active configuration window.
<code>\$\$report_entry_verification()</code>	Reports the integrity of entries in the active configuration.
<code>\$report_entry_verification()</code>	Reports the integrity of entries in the active configuration.
<code>\$\$report_global_status()</code>	Reports any errors currently on the global toolkit status stack to the transcript window.
<code>\$report_object_info()</code>	Displays information about the selected design object.
<code>\$report_reference_info()</code>	Displays information about the selected reference.
<code>\$report_tool_info()</code>	Displays information about the selected tool.
<code>\$report_type_info()</code>	Displays the properties of the selected type.
<code>\$report_version_info()</code>	Displays information about the selected version.
<code>\$\$resolve_path()</code>	Resolves a specified pathname into a hard pathname that does not have a symbolic link in its path.
<code>\$\$revert_version()</code>	Deletes the current version of the specified design object and makes the most recent version the new current version.
<code>\$revert_version()</code>	Deletes the current version of the selected design object and makes the most recent version the new current version.
<code>\$\$salvage_object()</code>	Attempts to repair a design object that has existing session data or malformed locks resulting from an application failure.
<code>\$salvage_object()</code>	Attempts to repair a design object that has existing session data or malformed locks resulting from an application failure.
<code>\$\$save_configuration()</code>	Writes the active configuration to the disk and increments the version number of the configuration object.

Table 7-1. Pyxis Project Manager Function Summary

Function	Description
<code>\$save_configuration()</code>	Writes the entries in the active configuration window to disk and increments the version number of the configuration object.
<code>\$\$save_configuration_as()</code>	Saves the active configuration under a new name.
<code>\$save_configuration_as()</code>	Saves the entries in the active configuration under a new configuration object name.
<code>\$\$save_object()</code>	Saves the specified locked design object.
<code>\$save_toolbox_search_path()</code>	Saves the toolbox search path.
<code>\$search()</code>	Searches the active window for the specified pattern.
<code>\$search_again()</code>	Repeats the search of the most recent search pattern specified.
<code>\$select_all()</code>	Selects all design objects in the active window.
<code>\$select_by_library()</code>	Selects specific parts libraries in a configuration.
<code>\$select_by_name()</code>	Selects a design object based on its name.
<code>\$select_by_type()</code>	Selects a design object based on its type.
<code>\$select_config_entry()</code>	Selects a configuration entry.
<code>\$select_object()</code>	Selects a design object.
<code>\$select_reference()</code>	Selects a reference.
<code>\$select_tool()</code>	Selects a tool.
<code>\$select_toolbox()</code>	Selects a toolbox.
<code>\$select_trash_object()</code>	Selects a trash object.
<code>\$select_version()</code>	Selects a design object version.
<code>\$set_build_rules()</code>	Sets the build rules for the selected primary entries.
<code>\$\$set_location_map_entry()</code>	Overrides a location map entry in memory.
<code>\$\$set_monitor_flag()</code>	Changes a single attribute of the configuration monitoring setup.
<code>\$\$set_monitor_verbosity()</code>	Sets the configuration monitor verbosity level.
<code>\$\$set_object_path_filter()</code>	Sets the value of the object path filter for the specified primary entry.

Table 7-1. Pyxis Project Manager Function Summary

Function	Description
<code>\$\$set_object_property()</code>	Sets the property of the specified design object.
<code>\$\$set_object_property_filter()</code>	Sets the value of the object property filter for the specified primary entry.
<code>\$\$set_object_type_filter()</code>	Sets the value of the object type filter for the specified primary entry.
<code>\$\$set_protection()</code>	Sets the protection of the specified design object.
<code>\$\$set_protection_numeric()</code>	Sets the protection of the specified design object by using an integer.
<code>\$\$set_reference_property()</code>	Sets the property of the specified reference.
<code>\$\$set_reference_property_filter()</code>	Sets the value of the reference property filter for the specified primary entry.
<code>\$\$set_reference_property_handle()</code>	Sets the property of the specified reference by using the target reference handle.
<code>\$\$set_reference_traversal()</code>	Sets the reference traversal of the specified primary entry.
<code>\$set_subinvoke_mode()</code>	Sets the tool invocation method to either send transcript messages to a new shell window or to save transcript messages to a log file.
<code>\$\$set_target_path()</code>	Sets the target path of the specified configuration entries.
<code>\$set_target_path()</code>	Sets the target path of selected configuration entries in the active configuration window.
<code>\$set_toolbox_search_path()</code>	Sets the order of the toolbox search path.
<code>\$\$set_version_depth()</code>	Sets the version depth of specified design object.
<code>\$\$set_version_property()</code>	Sets the property of the specified version.
<code>\$\$set_working_directory()</code>	Changes the value of the MGC working directory.
<code>\$set_working_directory()</code>	Changes the value of the MGC working directory.
<code>\$setup_filter_active()</code>	Specifies filters that determine which objects are displayed in the active navigator.
<code>\$setup_filter_all()</code>	Specifies filters that determine which objects are displayed in all navigators that you open after executing this function.

Table 7-1. Pyxis Project Manager Function Summary

Function	Description
<code>\$setup_default_editor()</code>	Sets the default editor.
<code>\$setup_iconic_window_layout()</code>	Specifies the layout features in iconic windows.
<code>\$setup_invoke_tool()</code>	Sets the active tool viewpoint.
<code>\$\$setup_monitor()</code>	Specifies the setup of configuration monitoring facilities.
<code>\$setup_monitor()</code>	Specifies the setup of the configuration monitoring facilities.
<code>\$setup_session_defaults()</code>	Specifies the session defaults for the Pyxis Project Manager graphical interface.
<code>\$setup_startup_windows()</code>	Specifies which windows are opened at invocation.
<code>\$show_component_hierarchy()</code>	Displays the hierarchy of a design object in a component or IC design hierarchy window.
<code>\$\$show_location_map()</code>	Displays, in a read-only window, the absolute path of the current location map, and the value of its entries in location map format.
<code>\$show_location_map()</code>	Displays, in a read-only window, the absolute path of the current location map, and the value of its entries.
<code>\$show_monitor()</code>	Makes the configuration monitor window visible.
<code>\$show_n_levels()</code>	Updates the hierarchy list to show components underneath the current object to the specified hierarchical depth.
<code>\$show_references()</code>	Displays the references of the selected design object.
<code>\$show_versions()</code>	Displays the versions of the selected design object.
<code>\$trash_object()</code>	Moves the selected design object to the trash window.
<code>\$\$unfreeze_configuration()</code>	Unfreezes all design object versions in the active configuration.
<code>\$unfreeze_configuration()</code>	Unfreezes all design object versions in the active configuration window.

Table 7-1. Pyxis Project Manager Function Summary

Function	Description
<code>\$\$unfreeze_version()</code>	Unfreezes a frozen version of the specified design object.
<code>\$unfreeze_version()</code>	Unfreezes a frozen version of the selected design object.
<code>\$\$unlock_configuration()</code>	Unlocks all design object versions in the active configuration.
<code>\$unlock_configuration()</code>	Unlocks all design object versions in the active configuration window.
<code>\$\$unlock_object()</code>	Unlocks the specified design object.
<code>\$unselect_all()</code>	Unselects all selected design objects.
<code>\$unselect_by_name()</code>	Unselects a design object based on its name.
<code>\$unselect_by_type()</code>	Unselects a design object based on its type.
<code>\$unselect_config_entry()</code>	Unselects all selected configuration entries.
<code>\$unselect_object()</code>	Unselects the selected design object.
<code>\$unselect_reference()</code>	Unselects the selected reference.
<code>\$unselect_tool()</code>	Unselects the selected tool.
<code>\$unselect_toolbox()</code>	Unselects the selected toolbox.
<code>\$unselect_trash_object()</code>	Unselects the selected trash objects.
<code>\$unselect_version()</code>	Unselects the selected version.
<code>\$untrash_object()</code>	Removes the selected design object from the trash window.
<code>\$update_window()</code>	Redraws the currently active window
<code>\$view_by_icon()</code>	Places the current navigator in iconic-viewing mode.
<code>\$view_by_name()</code>	Places the current navigator in list-viewing mode.
<code>\$viewContainmentHierarchy()</code>	Displays the containment hierarchy of all entries in the active configuration window.
<code>\$viewPrimaryHierarchy()</code>	Displays the primary entries in the active configuration window.
<code>\$viewSecondaryEntries()</code>	Displays the secondary entries in the active configuration window.

Table 7-1. Pyxis Project Manager Function Summary

Function	Description
\$view_toolboxes()	Changes the viewing mode of the tools window to display the toolboxes in the toolbox search path.
\$view_tools()	Changes the viewing mode of the tools window to display the current set of tool icons.
\$write_default_startup_file()	Saves the values of the current session settings to the default startup file.
\$\$writeln_monitor()	Writes the specified text string to the configuration's monitor window.

Interactive Function Summary

You can use these functions interactively during a Pyxis Project Manager session.

You can also use these functions in startup scripts to achieve a common look-and-feel among tools.

Table 7-2. Interactive Function Summary

Function	Description
\$add_configuration_entry()	Adds the current version of a design object to the active configuration window.
\$add_container()	Creates a basic container design object with an attribute file.
\$add_directory()	Creates a directory without an attribute file.
\$add_object_property()	Adds a property to the selected design object.
\$add_reference()	Creates a reference from the selected design object to a target design object.
\$add_reference_property()	Adds a property to the selected reference.
\$add_toolbox()	Appends the specified toolbox to the current toolbox search path.
\$add_versions()	Adds the specified versions of the selected design object to the active configuration window.
\$browse_for_object()	Brings up an object browser dialog.
\$build_configuration()	Adds secondary entries to the active configuration window, based on the build rules of the primary entries.

Table 7-2. Interactive Function Summary (cont.)

Function	Description
<code>\$change_configuration_references()</code>	Changes the references of each entry in the active configuration window.
<code>\$change_location_map_entry()</code>	Overrides a location map entry in memory.
<code>\$change_object_name()</code>	Renames the specified design object.
<code>\$change_object_property()</code>	Changes a property value of the selected design object.
<code>\$change_object_references()</code>	Changes the references of the selected design object.
<code>\$change_protection()</code>	Changes the protection of the selected design object.
<code>\$change_reference_property()</code>	Changes a property of the selected reference.
<code>\$change_reference_state()</code>	Changes the state of the selected reference.
<code>\$change_version_depth()</code>	Changes the version depth of the selected design object.
<code>\$check_references()</code>	Provides a simplified interface to check and fix the broken references of the selected design objects.
<code>\$check_registries()</code>	Checks the \$MGC_HOME/registry directory.
<code>\$close_window()</code>	Closes the active window.
<code>\$convert_configuration_references()</code>	Automatically converts reference pathnames on all objects in the configuration from hard reference pathnames to soft reference pathnames using the active location map.
<code>\$convert_object_references()</code>	Automatically converts the reference pathnames of selected design objects from hard to soft using the current location map.
<code>\$copy_configuration()</code>	Copies all design object versions in the active configuration window to a target location.
<code>\$copy_object()</code>	Copies the selected design objects to a target location.
<code>\$copy_version()</code>	Copies a single selected version of the design object to a target location.
<code>\$delete_configuration()</code>	Deletes all design object versions in the active configuration window from the disk.
<code>\$delete_excess_versions()</code>	Deletes all but a specified number of versions of the selected design object(s).

Table 7-2. Interactive Function Summary (cont.)

Function	Description
<code>\$delete_object()</code>	Deletes the selected design object.
<code>\$delete_object_property()</code>	Deletes the specified property from the selected design object.
<code>\$delete_reference()</code>	Deletes the selected reference from the source design object.
<code>\$delete_reference_property()</code>	Deletes the specified property from the selected reference.
<code>\$delete_version()</code>	Deletes the selected design object version.
<code>\$edit_file()</code>	Opens a file in edit or read-only mode.
<code>\$empty_trash()</code>	Deletes the contents of the trash window.
<code>\$explore_contents()</code>	Navigates down one level of the containment hierarchy.
<code>\$explore_parent()</code>	Navigates up one level of the containment or reference network.
<code>\$explore_reference_parent()</code>	Navigates to and displays the contents of the parent directory of the referenced object.
<code>\$explore_references()</code>	Navigates down one level of the reference network.
<code>\$export_location_map()</code>	The function <code>\$export_location_map()</code> causes the current location map to be written to a tmp file. This function also shows which tmp files can be removed after executing <code>\$show_location_map()</code> .
<code>\$find_references()</code>	Finds the references for the selected part
<code>\$freeze_configuration()</code>	Freezes all design object versions in the active configuration window and creates a new version of the configuration.
<code>\$freeze_version()</code>	Freezes the selected design object version.
<code>\$get_area_selected_objects()</code>	Returns the pathnames and types of the selected design objects in the navigator window.
<code>\$get_default_tool()</code>	Returns the default tool for the specified type.
<code>\$get_navigator_directory()</code>	Returns the soft pathname of the current navigator directory.
<code>\$get_navigator_directory_hard()</code>	Returns the hard pathname of the current navigator directory.
<code>\$get_toolbox_search_path()</code>	Returns the current toolbox search path.

Table 7-2. Interactive Function Summary (cont.)

Function	Description
<code>\$goto_directory()</code>	Navigates to the specified directory.
<code>\$hide_secondary_entries()</code>	Hides the secondary entries in the active configuration window.
<code>\$hide_monitor()</code>	Hides the configuration's monitor window.
<code>\$list_references()</code>	List the all references for a component.
<code>\$load_registry()</code>	Loads a type registry into the current session's type manager.
<code>\$lock_configuration()</code>	Locks all design object versions in the active configuration window.
<code>\$maintain_hierarchy()</code>	Maintains a parts directory hierarchy upon release.
<code>\$move_object()</code>	Moves the selected design object to a target location.
<code>\$open_configuration_window()</code>	Opens a new or existing configuration object.
<code>\$open_navigator()</code>	Opens a navigator.
<code>\$open_object()</code>	Opens the selected design object.
<code>\$open_read_only_editor()</code>	Invokes the read-only ASCII editor on the selected object.
<code>\$open_session_monitor()</code>	Makes the session monitor window visible.
<code>\$open_tool()</code>	Invokes the selected tool.
<code>\$open_tools_window()</code>	Opens a tools window.
<code>\$open_trash_window()</code>	Opens a trash window.
<code>\$open_types_window()</code>	Displays the types currently loaded in the Pyxis Project Manager.
<code>\$read_map()</code>	Reads the ASCII location map file into memory.
<code>\$release_configuration()</code>	Releases the design object versions in the active configuration window to a target location.
<code>\$release_object()</code>	Releases the current version of the selected objects to the specified destination directory.
<code>\$remove_configuration_entry()</code>	Removes an entry from the active configuration window.
<code>\$remove_toolbox()</code>	Removes a toolbox from the toolbox search path.

Table 7-2. Interactive Function Summary (cont.)

Function	Description
<code>\$report_configuration_info()</code>	Displays the status of the active configuration window.
<code>\$report_configuration_references()</code>	Reports the references of entries in the active configuration window.
<code>\$report_entry_info()</code>	Displays the status of the selected entries in the active configuration window.
<code>\$report_entry_verification()</code>	Reports the integrity of entries in the active configuration.
<code>\$report_object_info()</code>	Displays information about the selected design object.
<code>\$report_reference_info()</code>	Displays information about the selected reference.
<code>\$report_tool_info()</code>	Displays information about the selected tool.
<code>\$report_type_info()</code>	Displays the properties of the selected type.
<code>\$report_version_info()</code>	Displays information about the selected version.
<code>\$revert_version()</code>	Deletes the current version of the selected design object and makes the most recent version the new current version.
<code>\$salvage_object()</code>	Attempts to repair a design object that has existing session data or malformed locks resulting from an application failure.
<code>\$save_configuration()</code>	Writes the entries in the active configuration window to disk and increments the version number of the configuration object.
<code>\$save_configuration_as()</code>	Saves the entries in the active configuration under a new configuration object name.
<code>\$save_toolbox_search_path()</code>	Saves the toolbox search path.
<code>\$search()</code>	Searches the active window for the specified pattern.
<code>\$search_again()</code>	Repeats the search of the most recent search pattern specified.
<code>\$select_all()</code>	Selects all design objects in the active window.
<code>\$select_by_name()</code>	Selects a design object based on its name.
<code>\$select_by_library()</code>	Selects specific parts libraries in a configuration.
<code>\$select_by_type()</code>	Selects a design object based on its type.

Table 7-2. Interactive Function Summary (cont.)

Function	Description
\$select_config_entry()	Selects a configuration entry.
\$select_object()	Selects a design object.
\$select_reference()	Selects a reference.
\$select_tool()	Selects a tool.
\$select_toolbox()	Selects a toolbox.
\$select_trash_object()	Selects a trash object.
\$select_version()	Selects a design object version.
\$set_build_rules()	Sets the build rules for the selected primary entries.
\$set_target_path()	Sets the target path of selected configuration entries in the active configuration window.
\$set_toolbox_search_path()	Sets the order of the toolbox search path.
\$set_working_directory()	Changes the value of the MGC working directory.
\$setup_filter_active()	Specifies filters that determine which objects are displayed in the active navigator.
\$setup_filter_all()	Specifies filters that determine which objects are displayed in all navigators that you open after executing this function.
\$setup_default_editor()	Sets the default editor.
\$setup_iconic_window_layout()	Specifies the layout features in iconic windows.
\$setup_monitor()	Specifies the setup of the configuration monitoring facilities.
\$setup_session_defaults()	Specifies the session defaults for the Pyxis Project Manager graphical interface.
\$setup_startup_windows()	Specifies which windows are opened at invocation.
\$show_location_map()	Displays, in a read-only window, the absolute path of the current location map, and the value of its entries in location map format.
\$show_references()	Displays the references of the selected design object.
\$show_monitor()	Makes the configuration monitor window visible.

Table 7-2. Interactive Function Summary (cont.)

Function	Description
<code>\$show_versions()</code>	Displays the versions of the selected design object.
<code>\$trash_object()</code>	Moves the selected design object to the trash window.
<code>\$unfreeze_configuration()</code>	Unfreezes all design object versions in the active configuration window.
<code>\$unfreeze_version()</code>	Unfreezes a frozen version of the selected design object.
<code>\$unlock_configuration()</code>	Unlocks all design object versions in the active configuration window.
<code>\$unselect_all()</code>	Unselects all selected design objects.
<code>\$unselect_by_name()</code>	Unselects a design object based on its name.
<code>\$unselect_by_type()</code>	Unselects a design object based on its type.
<code>\$unselect_config_entry()</code>	Unselects all selected configuration entries.
<code>\$unselect_object()</code>	Unselects the selected design object.
<code>\$unselect_reference()</code>	Unselects the selected reference.
<code>\$unselect_tool()</code>	Unselects the selected tool.
<code>\$unselect_toolbox()</code>	Unselects the selected toolbox.
<code>\$unselect_trash_object()</code>	Unselects the selected trash objects.
<code>\$unselect_version()</code>	Unselects the selected version.
<code>\$untrash_object()</code>	Removes the selected design object from the trash window.
<code>\$update_window()</code>	Updates the currently active window with the most current information.
<code>\$view_by_icon()</code>	Places the current navigator in iconic-viewing mode.
<code>\$view_by_name()</code>	Places the current navigator in list-viewing mode.
<code>\$viewContainmentHierarchy()</code>	Displays the containment hierarchy of all entries in the active configuration window.
<code>\$viewPrimaryHierarchy()</code>	Displays the primary entries in the active configuration window.
<code>\$viewSecondaryEntries()</code>	Displays the secondary entries in the active configuration window.

Table 7-2. Interactive Function Summary (cont.)

Function	Description
\$view_toolboxes()	Changes the viewing mode of the tools window to display the toolboxes in the toolbox search path.
\$view_tools()	Changes the viewing mode of the tools window to display the current set of tool icons.
\$write_default_startup_file()	Saves the values of the current session settings to the default startup file.

Design Object Toolkit Function Summary

A *toolkit* is a set of functions that you use to create sophisticated, batch mode scripts.

The design object toolkit functions allow you to manipulate design objects. They are available from any Pyxis Project Manager window and from the tool viewpoint scripts.

Table 7-3. Design Object Toolkit Function Summary

Function	Description
\$\$add_directory()	Creates a container design object without an attribute file.
\$\$add_reference()	Creates a reference from the specified source design object to the specified target design object.
\$\$clear_global_status()	Clears the global toolkit status stack.
\$\$close_versioned_object()	Closes a versioned design object that encapsulates data from a non-Mentor Graphics application.
\$\$create_versioned_object()	Creates a versioned design object that encapsulates data from a non-Mentor Graphics application.
\$delete_excess_versions()	Deletes all but a specified number of versions of the selected design object(s).
\$\$delete_object()	Deletes the specified design object.
\$\$delete_object_property()	Deletes the specified property from the specified design object.
\$\$delete_reference()	Deletes the specified reference from the specified source design object.
\$\$delete_reference_handle()	Deletes a reference from the specified design object by using the reference target handle.
\$\$delete_reference_property()	Deletes a property from the specified reference.

Table 7-3. Design Object Toolkit Function Summary (cont.)

Function	Description
<code>\$\$delete_reference_property_handle()</code>	Deletes a property from the specified reference by using the reference target handle.
<code>\$\$delete_version()</code>	Deletes the specified version of the specified design object.
<code>\$\$delete_version_property()</code>	Deletes a property from the specified version.
<code>\$\$fix_relative_path()</code>	Generates an absolute pathname from a relative pathname.
<code>\$\$freeze_version()</code>	Freezes a version of the specified design object.
<code>\$\$get_container_contents()</code>	Returns the name, type, and version of the objects in the specified container.
<code>\$\$get_date_last_modified()</code>	Returns the date that the specified design object was last saved to disk.
<code>\$\$get_fileset_members()</code>	Returns the fileset members of the specified design object.
<code>\$\$get_hard_name()</code>	Returns the hard pathname equivalent of any pathname.
<code>\$\$get_location_map()</code>	Returns the absolute path of the current location map, and the values of its entries.
<code>\$\$get_object_current_version()</code>	Returns the current version of the specified design object.
<code>\$\$get_object_parent_path()</code>	Returns the pathname of the parent of the specified design object.
<code>\$\$get_object_properties()</code>	Returns the properties of the specified design object.
<code>\$\$get_object_property_value()</code>	Returns a property value of the specified design object.
<code>\$\$get_object_protection()</code>	Returns the protection status of the specified design object.
<code>\$\$get_object_references()</code>	Returns the references of the specified design object.
<code>\$\$get_object_type()</code>	Returns the design object type of the specified design object.
<code>\$\$get_object_versions()</code>	Returns the versions of the specified design object.
<code>\$\$get_reference_properties()</code>	Returns the properties of the specified reference.
<code>\$\$get_reference_properties_handle()</code>	Returns the properties of the specified reference by using the target reference handle.

Table 7-3. Design Object Toolkit Function Summary (cont.)

Function	Description
<code>\$\$get_soft_name()</code>	Returns the soft pathname equivalent of a pathname.
<code>\$\$get_status_code()</code>	Returns the error message code at the top of the status stack.
<code>\$\$get_status_code_stack()</code>	Returns all of the error message codes on the status code stack.
<code>\$\$get_status_messages()</code>	Returns the error messages on the status stack.
<code>\$\$get_type_properties()</code>	Returns all the properties associated with that type.
<code>\$\$get_type_property_value()</code>	Returns the value of the property type.
<code>\$\$get_version_depth()</code>	Returns the version depth of the specified design object.
<code>\$\$get_version_properties()</code>	Returns the properties of the specified design object version.
<code>\$\$get_working_directory()</code>	Returns the value of the MGC working directory.
<code>\$\$handle_map_error()</code>	Lets the user determine how to proceed when an error occurs in reading a location map.
<code>\$\$has_object_property()</code>	Checks if a property exists on the specified design object.
<code>\$\$has_reference_property()</code>	Checks if a property exists on the specified reference.
<code>\$\$has_reference_property_handle()</code>	Checks if a property exists on the specified reference by using the target reference handle.
<code>\$\$is_container()</code>	Checks if the specified design object is a container.
<code>\$\$is_directory()</code>	Checks if the specified design object is a directory.
<code>\$\$is_object_released()</code>	Checks if the specified design object is in a released state.
<code>\$\$is_object_versioned()</code>	Checks if the specified design object is versioned.
<code>\$\$is_read_protected()</code>	Checks if the specified design object can be opened for reading.
<code>\$\$is_relative_path()</code>	Indicates if the specified pathname is a relative pathname.
<code>\$\$is_type_versioned()</code>	Checks if the type is versioned.
<code>\$\$is_writable()</code>	Checks if the current process can write to the specified design object.

Table 7-3. Design Object Toolkit Function Summary (cont.)

Function	Description
<code>\$\$is_write_protected()</code>	Checks if the specified design object can be opened for writing.
<code>\$\$lock_object()</code>	Locks the specified design object.
<code>\$\$monitor_global_status()</code>	Reports any errors currently on the global toolkit status stack.
<code>\$\$object_complete()</code>	Checks if the specified design object is complete.
<code>\$\$object_exists()</code>	Checks if the specified design object exists.
<code>\$\$open_tool()</code>	Invokes the specified tool.
<code>\$\$open_versioned_object()</code>	Opens a versioned design object that encapsulates data from a non-Mentor Graphics application.
<code>\$\$read_map()</code>	Reads the ASCII location map file into memory.
<code>\$\$report_global_status()</code>	Reports any errors currently on the global toolkit status stack to the transcript window.
<code>\$\$resolve_path()</code>	Resolves a specified pathname into a hard pathname that does not have a symbolic link in its path.
<code>\$\$revert_version()</code>	Deletes the current version of the specified design object and makes the most recent version the new current version.
<code>\$\$salvage_object()</code>	Attempts to repair a design object that has existing session data or malformed locks resulting from an application failure.
<code>\$\$save_object()</code>	Saves the specified locked design object.
<code>\$\$set_location_map_entry()</code>	Overrides a location map entry in memory.
<code>\$\$set_object_property()</code>	Sets the property of the specified design object.
<code>\$\$set_protection()</code>	Sets the protection of the specified design object.
<code>\$\$set_protection_numeric()</code>	Sets the protection of the specified design object by using an integer.
<code>\$\$set_reference_property()</code>	Sets the property of the specified reference.
<code>\$\$set_reference_property_handle()</code>	Sets the property of the specified reference by using the target reference handle.
<code>\$\$set_version_depth()</code>	Sets the version depth of specified design object.
<code>\$\$set_version_property()</code>	Sets the property of the specified version.
<code>\$\$set_working_directory()</code>	Changes the value of the MGC working directory.

Table 7-3. Design Object Toolkit Function Summary (cont.)

Function	Description
\$\$show_location_map()	Displays, in a read-only window, the absolute path of the current location map, and the value of its entries.
\$\$unfreeze_version()	Unfreezes a frozen version of the specified design object.
\$\$unlock_object()	Unlocks the specified design object.

Configuration Management Toolkit Function Summary

The configuration toolkit functions allow you to manipulate configurations, and to move and copy design objects. They are available from any Pyxis Project Manager window and from the tool viewpoint scripts.

Table 7-4. Configuration Toolkit Function Summary

Function	Description
<code>\$\$add_configuration_entry()</code>	Adds the specified design object version to the active configuration.
<code>\$\$add_container()</code>	Creates a basic container design object with an attribute file.
<code>\$\$build_configuration()</code>	Adds secondary entries to the active configuration, based on the build rules of the primary entries.
<code>\$\$change_configuration_references()</code>	Changes the references of each entry in the active configuration.
<code>\$\$change_object_name()</code>	Renames the specified design object.
<code>\$\$change_object_references()</code>	Changes the references of the specified design object.
<code>\$\$clear_entry_filter()</code>	Resets all filters of a primary entry's build rules.
<code>\$\$clear_monitor()</code>	Clears all text from the configuration's monitor window.
<code>\$\$close_configuration()</code>	Removes the lock on the active configuration.
<code>\$\$convert_configuration_references()</code>	Converts the references of each entry in the active configuration.
<code>\$\$convert_object_references()</code>	Converts the references of the specified design object(s).
<code>\$\$copy_configuration()</code>	Copies all design object versions in the active configuration, to a target location.
<code>\$\$copy_object()</code>	Copies the specified design object to a target location.
<code>\$\$create_configuration()</code>	Creates a configuration object.
<code>\$\$delete_configuration()</code>	Deletes all design object versions in the active configuration from the disk.
<code>\$\$duplicate_object()</code>	Duplicates a design object.
<code>\$\$freeze_configuration()</code>	Freezes all design object versions in the active configuration.

Table 7-4. Configuration Toolkit Function Summary (cont.)

Function	Description
<code>\$\$get_children()</code>	Returns the children of a configuration entry.
<code>\$\$get_configuration_entries()</code>	Returns all entries in the active configuration.
<code>\$\$get_configuration_path()</code>	Returns the soft pathname of the active configuration.
<code>\$\$get_entry_version()</code>	Returns the version number of the target object of a current entry in the active configuration.
<code>\$\$get_monitor_error_count()</code>	Returns the number of errors processed by the monitor.
<code>\$\$get_monitor_flag()</code>	Returns the value of the specified configuration monitoring attribute.
<code>\$\$get_monitor_verbosity()</code>	Returns the value of the specified configuration monitoring attribute.
<code>\$\$get_monitor_warning_count()</code>	Returns the number of warnings processed by the monitor.
<code>\$\$get_object_path_filter()</code>	Returns the value of the object path filter for the specified primary entry.
<code>\$\$get_object_property_filter()</code>	Returns the value of the object property filter for the specified primary entry.
<code>\$\$get_object_type_filter()</code>	Returns the value of the object type filter for the specified primary entry.
<code>\$\$get_parent_entry()</code>	Returns the parent of the specified configuration entry.
<code>\$\$get_primaries()</code>	Returns the primary entries in the active configuration.
<code>\$\$get_reference_property_filter()</code>	Returns the value of the reference property filter for the specified primary entry.
<code>\$\$get_reference_traversal()</code>	Returns the reference traversal of the specified primary entry.
<code>\$\$get_secondaries()</code>	Returns the secondary entries of the specified primary entry.
<code>\$\$get_target_path()</code>	Returns the target path of the specified primary configuration entry.
<code>\$\$is_build_consistent()</code>	Checks if the active configuration is consistent.
<code>\$\$is_build_valid()</code>	Checks if the active configuration is valid.

Table 7-4. Configuration Toolkit Function Summary (cont.)

Function	Description
<code>\$\$is_configuration_edited()</code>	Checks if the active configuration has been changed and needs to be saved.
<code>\$\$is_configuration_frozen()</code>	Checks if the active configuration is frozen.
<code>\$\$is_configuration_locked()</code>	Checks if the active configuration is locked.
<code>\$\$is_entry_container()</code>	Checks if the specified configuration entry is a container.
<code>\$\$is_entry_fixed()</code>	Checks if the specified configuration entry is a fixed entry.
<code>\$\$is_entry_primary()</code>	Checks if the specified configuration entry is a primary entry.
<code>\$\$is_entry_retargetable()</code>	Checks if the specified configuration entry is re-targetable.
<code>\$\$lock_configuration()</code>	Locks all design object versions in the active configuration.
<code>\$\$move_object()</code>	Moves the specified design objects to a target location.
<code>\$\$open_configuration()</code>	Opens the specified configuration object.
<code>\$\$prune_design_hierarchy()</code>	Prunes off back versions of a design object.
<code>\$\$release_configuration()</code>	Releases all design object versions in the active configuration to a target location.
<code>\$\$release_object()</code>	Releases the current version of the specified objects to the specified destination directory.
<code>\$\$remove_configuration_entry()</code>	Removes the specified entry from the active configuration.
<code>\$\$report_configuration_references()</code>	Reports the references of entries in the active configuration.
<code>\$\$report_entry_verification()</code>	Reports the integrity of entries in the active configuration.
<code>\$\$save_configuration()</code>	Writes the active configuration to the disk and increments the version number of the configuration object.
<code>\$\$save_configuration_as()</code>	Saves the active configuration under a new name.
<code>\$\$set_monitor_flag()</code>	Changes a single attribute of the configuration monitoring setup.
<code>\$\$set_monitor_verbosity()</code>	Sets the configuration monitor verbosity level.

Table 7-4. Configuration Toolkit Function Summary (cont.)

Function	Description
<code>\$\$set_object_path_filter()</code>	Sets the value of the object path filter for the specified primary entry.
<code>\$\$set_object_property_filter()</code>	Sets the value of the object property filter for the specified primary entry.
<code>\$\$set_object_type_filter()</code>	Sets the value of the object type filter for the specified primary entry.
<code>\$\$set_reference_property_filter()</code>	Sets the value of the reference property filter for the specified primary entry.
<code>\$\$set_reference_traversal()</code>	Sets the reference traversal of the specified primary entry.
<code>\$\$set_target_path()</code>	Sets the target path of the specified configuration entries.
<code>\$\$setup_monitor()</code>	Specifies the setup of configuration monitoring facilities.
<code>\$\$unfreeze_configuration()</code>	Unfreezes all design object versions in the active configuration.
<code>\$\$unlock_configuration()</code>	Unlocks all design object versions in the active configuration.
<code>\$\$writeln_monitor()</code>	Writes the specified text string to the configuration's monitor window.

Tool Viewpoint Function Summary

Some functions are available from inside the tool viewpoint qualification and termination scripts.



Tip: For more information on qualification and termination scripts, refer to “[Tool Invocation](#)” on page [102](#).

Table 7-5. Tool Viewpoint Function Summary

Function	Description
<code>\$get_object_pathname()</code>	Returns the absolute pathname of the specified design object during data-centered tool invocation, or the absolute pathname of the active tool viewpoint during tool-centered tool invocation.

Table 7-5. Tool Viewpoint Function Summary (cont.)

Function	Description
<code>\$get_object_type()</code>	Returns the type of the specified design object during data-centered tool invocation, or the type of the active tool viewpoint during tool-centered tool invocation.
<code>\$get_object_version()</code>	Returns the version number of the specified design object during data-centered tool invocation, or the version number of the active tool viewpoint during tool-centered tool invocation.
<code>\$get_subinvoke_mode()</code>	Returns the current tool invocation method.
<code>\$get_tool.pathname()</code>	Returns the absolute pathname of the active tool viewpoint during data-centered or tool-centered tool invocation.
<code>\$get_tool_script()</code>	Returns the absolute pathname of the active tool viewpoint's qualification or termination script.
<code>\$get_tool_type()</code>	Returns the type of the active tool viewpoint during data-centered or tool-centered tool invocation.
<code>\$invoke_bgd_tool()</code>	Invokes an executable file as a background process.
<code>\$invoke_tool()</code>	Opens a new shell window and invokes an executable file, when subinvoke_mode is set to send transcript messages to a new shell window.
<code>\$set_subinvoke_mode()</code>	Sets the tool invocation method to either send transcript messages to a new shell window or to save transcript messages to a log file.
<code>\$setup_invoke_tool()</code>	Sets the active tool viewpoint.

iDM Interactive Function Summary

The iDM interactive functions provide you with the ability to copy, move, delete, and change the references of design objects, as well as the ability to view the design hierarchy of component design objects from within Mentor Graphics EDA applications.

Table 7-6. iDM Interactive Function Summary

Function	Description
<code>\$change_design_object_references()</code>	Changes the references of the specified design objects.
<code>\$copy_design_object()</code>	Copies one or more design objects to a container.
<code>\$delete_design_object()</code>	Deletes the specified design objects.

Table 7-6. iDM Interactive Function Summary

Function	Description
<code>\$descend_hierarchy_one_level()</code>	Displays the next level of hierarchy directly beneath the selected component in a component hierarchy window.
<code>\$descend_hierarchy_specify_level()</code>	Displays the design hierarchy for a selected component in a component hierarchy window using specified level and filter settings.
<code>\$move_design_object()</code>	Moves one or more design objects to a new container.

iDM Toolkit Function Summary

The iDM toolkit functions are a set of functions that you use to write scripts to execute from within your EDA application. You can use these scripts to manipulate design objects from within your application in batch mode.

Table 7-7. iDM Toolkit Function Summary

Function	Description
\$\$change_design_object_references()	Changes the references of the specified design objects.
\$\$copy_design_object()	Copies one or more design objects to a container.
\$\$move_design_object()	Moves one or more design objects to a new container.
\$show_component_hierarchy()	Displays the hierarchy of a design object in a component or IC design hierarchy window.

Chapter 8

Language Flow

Pyxis Project Manager provides support for Verilog, System Verilog, Verilog AMS, Verilog A, VHDL, and VHDL AMS model compilation and simplifies creating, managing, and updating compiled hardware description language (HDL) libraries associated with a design hierarchy.

Pyxis Project Manager also provides support for creating and managing SPICE models.

Language Integration for Compilation	453
Model Registration or Symbol Generation	463
Default Symbol Layout	467
Registering a Model from a Source File	472
Registering All Models from a Source File	474
Registering a Model from a Compiled Library	475
Mapping or Unmapping Symbol Pins to the Model	477
Map Pins Dialog Box	477
Mapping or Unmapping Symbol Properties to the Model	480
Map Properties Dialog Box	480
Checking the Language Views	483
INI Mappings	484
Language Import	487

Language Integration for Compilation

The default compilation settings can be set per HDL type. These settings are applied to all HDL source files of that type within the current project or external library.

Compilation and registration settings can be managed for each language type at the project level; in addition, compilation settings can be managed for an individual file. By default, Pyxis Project Manager uses ADMS tools to compile all HDL.

The following features are supported by Pyxis Project Manager with respect to model compilation:

- Performing automatic library naming, placement, and mapping of compiled HDL libraries representing views within the Pyxis Project Manager's Project Navigator hierarchy.

- Differentiating and managing compiled HDL libraries automatically based on the Questa ADMS and ModelSim versions with which they were created.
- Supporting the mapping of compiled libraries located outside of the current hierarchy.

You can instruct the interface to use the default compilation settings using ADMS tools or specify alternate compilation settings. Depending on the selection level in the hierarchy, the appropriate dialog is opened for the Project Compilation Options or the Compilation Settings for the Language File. See [Figure 8-4](#) for details.

All the language flow operations in Pyxis Project Manager are available under the **Edit > Language** cascading menu. For more information on language flow functions in Pyxis Project Manager, refer to the [Pyxis Project Manager Reference Manual](#).

Prerequisites for Working with Language Files	454
Invoking the Language Editor from Pyxis Project Manager	454
Setup Preferences Dialog Box (Language Interface Panel)	457
Specifying Compilation Options for a Project	458
Specifying Alternate Compilation Settings for an HDL File	460
Compiling HDL source file(s)	462

Prerequisites for Working with Language Files

To work with any of the language files from the Pyxis Project Manager tool, the following prerequisites have to be enabled:

1. Your MGC_AMS_HOME environment variable must be set to the correct version.
2. If you are working on an empty project, you must create logic libraries before working on language files.
3. While not mandatory, you could also set up your preferences for Pyxis Project Manager as described in the [Setup Preferences Dialog Box \(Language Interface Panel\)](#) section.

Related Topics

[Setup Preferences Dialog Box \(Language Check Panel\)](#)

Invoking the Language Editor from Pyxis Project Manager

The Language Editor window in Pyxis Schematic is the default text editor for editing the source code for language files. Use this procedure for invoking the language editor from Pyxis Project Manager.

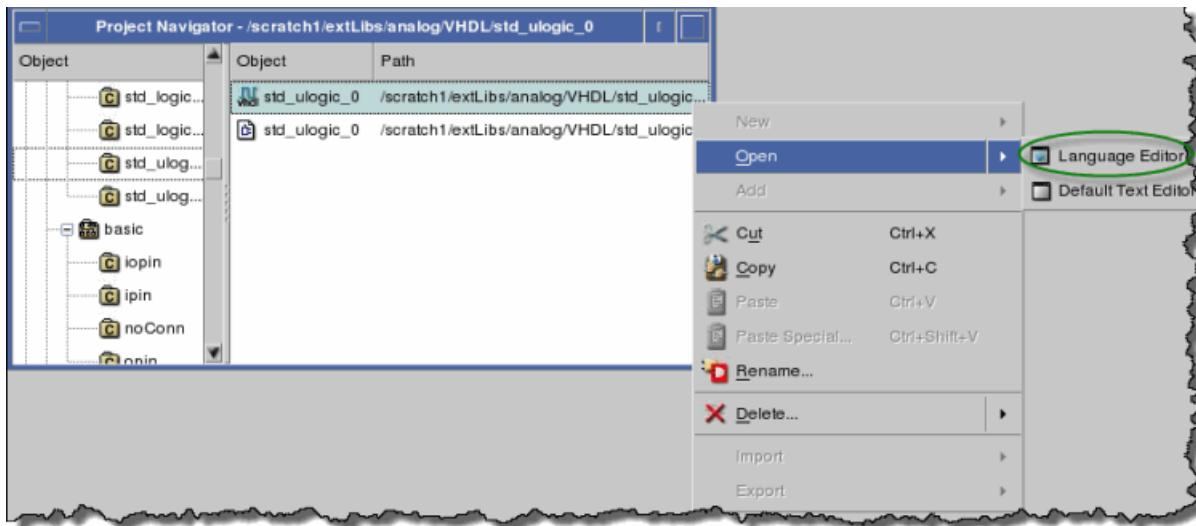
Prerequisites

- Pyxis Schematic has language files set up.
- The prerequisites specified in [Prerequisites for Working with Language Files](#) have been enabled.

Procedure

1. Double-click any language file in the project hierarchy.
- OR
2. Use the popup menu as follows:
 - a. Right-click any language file in the project hierarchy.
 - b. Select **Open > Language Editor** from the popup menu. See [Figure 8-1](#).

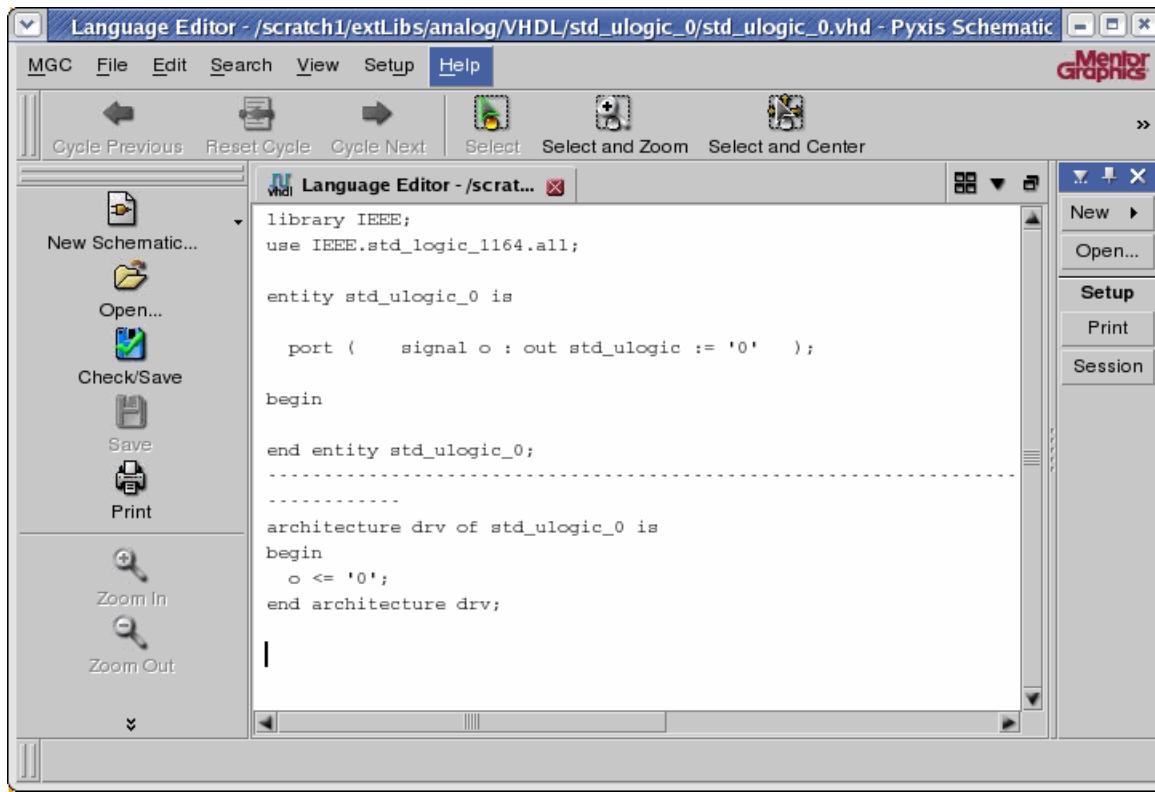
Figure 8-1. Invoking the Language Editor Using the Popup Menu



Results

Pyxis Schematic invokes with the source code displayed in the Language Editor. See the source code for the above language file in [Figure 8-2](#).

Figure 8-2. The Language Editor in Pyxis Schematic



Related Topics

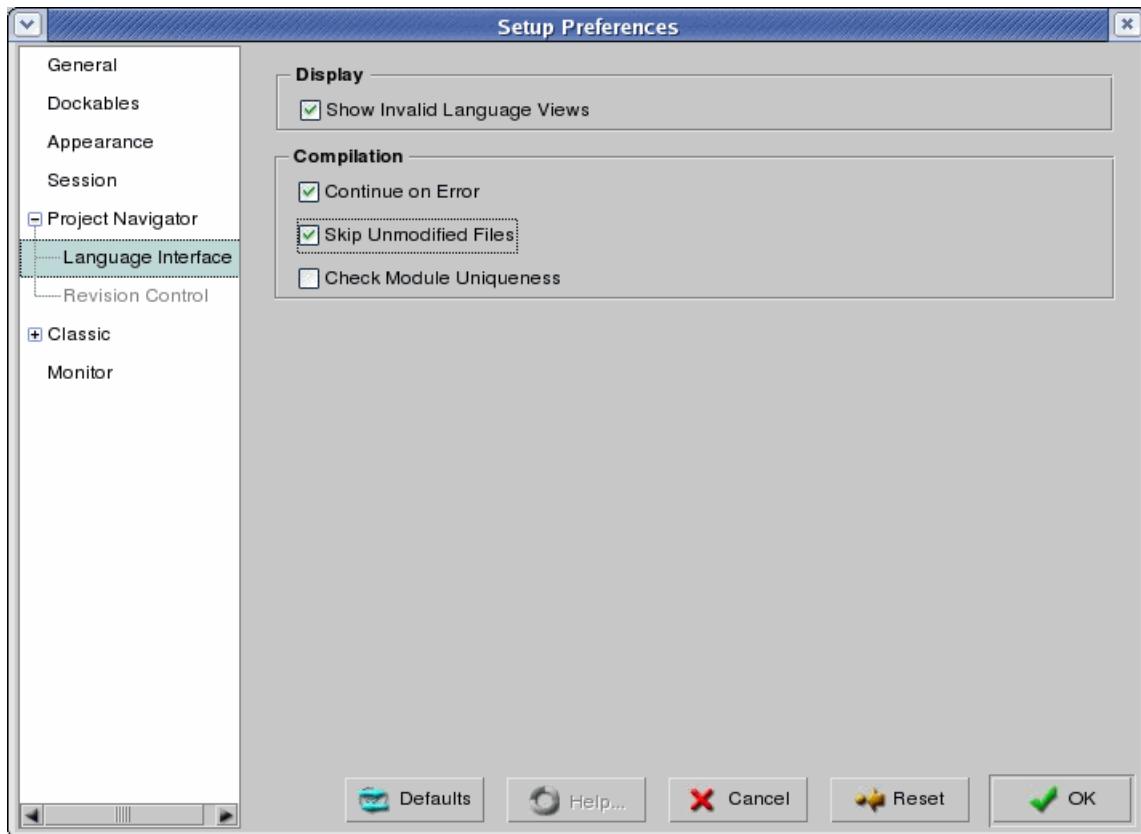
[Invoking the Language Editor From the Pyxis Schematic Application](#)

Setup Preferences Dialog Box (Language Interface Panel)

To access: Select the pulldown menu path **Setup > Preferences**; in the sidebar of the dialog that invokes, expand the **Project Navigator** option and select the **Language Interface** sub-option to display the panel shown in [Figure 8-3](#).

Use this panel to specify your selections for displaying invalid language views as well as the compilation settings for language files.

Figure 8-3. Setup Preferences Dialog Box (Language Interface Panel)



Fields

Table 8-1. Setup Preferences Dialog Box (Language Interface Panel) Settings

Field	Description
Show Invalid Language Views	When enabled, specifies to display (in red color) all the cases where the registered file does not match up with the compiled file. Hover over the file containing the error to obtain the solution for troubleshooting.
Continue on Error	When enabled, specifies that if the compilation process encounters an error and fails, then it should not stop but continue to compile other source files in the project.
Skip Unmodified Files	When enabled, specifies that when any container in the hierarchy is compiled, only the HDL objects in it that have been modified since the last compilation should be re-compiled.
Check Module Uniqueness	When enabled, specifies that if a design unit is imported bearing the same name as another in the active library or category, then an alert is to be generated before overwriting.

Related Topics

[\\$get_modules_uniqueness_check_pref\(\)](#)

[\\$set_modules_uniqueness_check_pref\(\)](#)

[\\$show_invalid_language_views\(\)](#)

[Setup Preferences Dialog Box \(Language Check Panel\)](#)

Specifying Compilation Options for a Project

Use this procedure to specify compilation settings for the project.

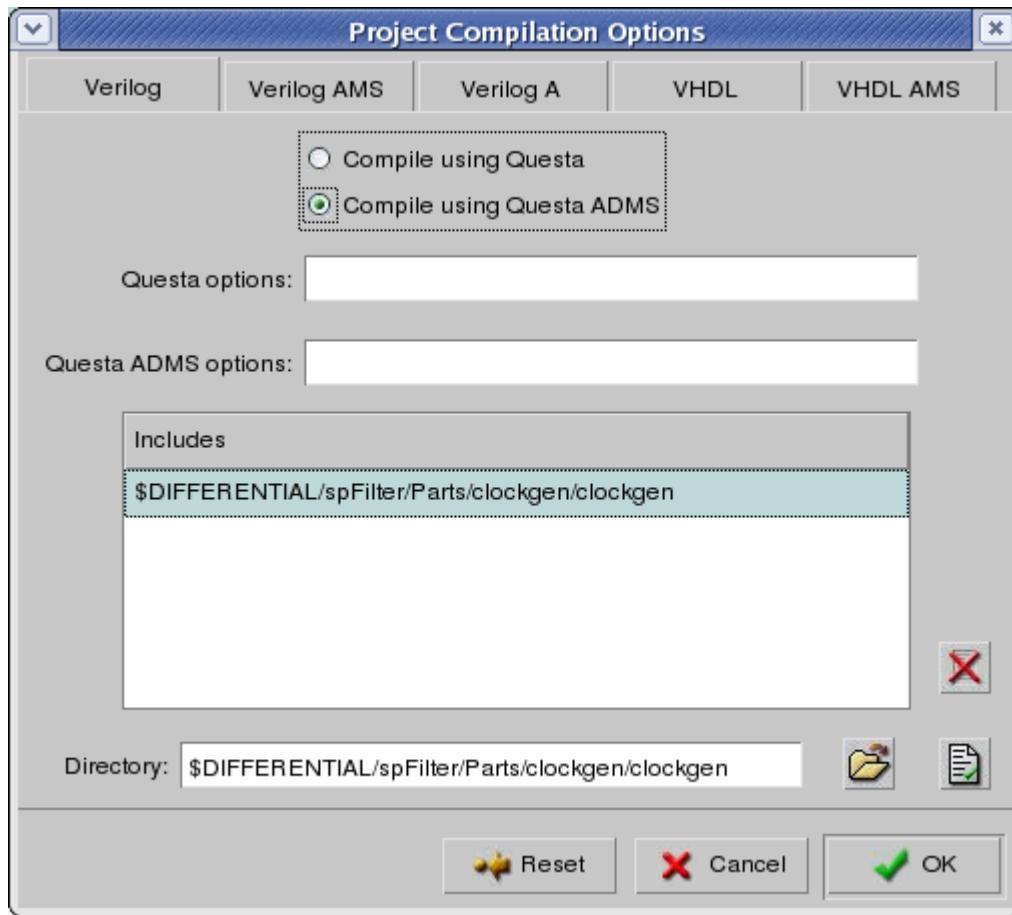
Prerequisites

- The Project Navigator window is open in Pyxis Project Manager.

Procedure

1. Select the project in the hierarchy.
2. Select the pulldown menu option **Edit > Language > Compilation Options**. This opens up a dialog box as shown in [Figure 8-4](#).

Figure 8-4. Project Compilation Options Dialog Box



3. Make your selections for each tab; the dialog box varies depending on the HDL language that you select. Some selections you can make are as follows:
 - a. To turn off ADMS compilation for Verilog and VHDL and compile for digital simulation only, select the **Compile using Questa** option.
 - b. To turn on Eldo compilation of Verilog A, select the Verilog A tab and type in a space-separated list of arguments in the **Eldo Options** text-entry box.
 - c. To specify the options for a particular compiler, type in a space-separated list of arguments in the **Questa options** or **Questa ADMS options** text-entry boxes.



Note

Pyxis Project Manager reserves the use of the ADMS compiler arguments "-work" and "-ms", and the Questa compiler argument "-work". Do not include these arguments in the "Questa ADMS options" or "Questa Options" string respectively.

4. Add Include paths using the **Directory** field and Pyxis Project Manager uses the paths listed in the **Includes** section of the dialog when invoking the specified compiler.

Related Topics

[Prerequisites for Working with Language Files](#)

[Specifying Alternate Compilation Settings for an HDL File](#)

[Specifying the Default Registration Settings](#)

Specifying Alternate Compilation Settings for an HDL File

Use this procedure to specify alternate compilation settings for an HDL file that overrides the project defaults.

Prerequisites

- The Project Navigator window is open in Pyxis Project Manager.

Procedure

1. Select the HDL file in the project for which to specify alternate compilation settings.
2. Select the pulldown menu option **Edit > Language > Compilation Options**. This invokes the “Compilation Options for <HDL_file_name>” dialog box.
3. Un-select the checkbox option “Use default options for hierarchy”.

The “Compilation Settings for <HDL_file_name>” dialog box invoked varies with respect to the type of HDL file selected, as shown in [Figure 8-5](#) (Verilog file) and [Figure 8-6](#) (VHDL file).

Figure 8-5. Compilation Options for <Verilog_file_name> Dialog Box

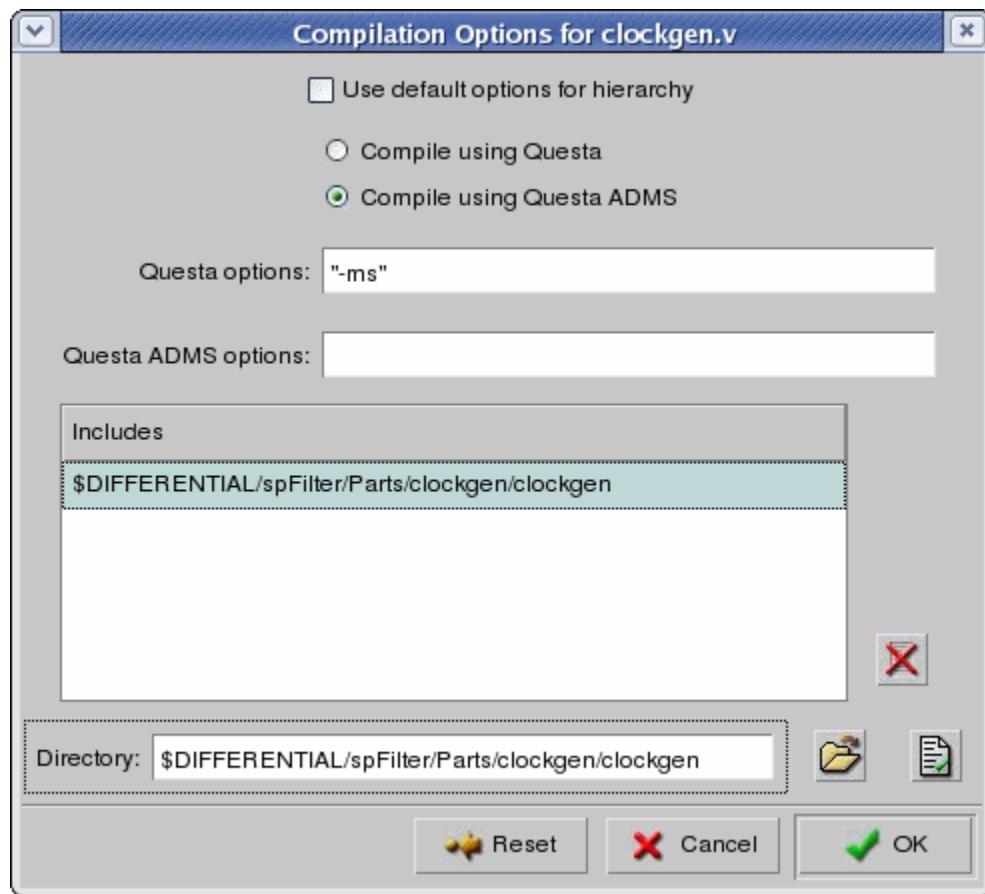
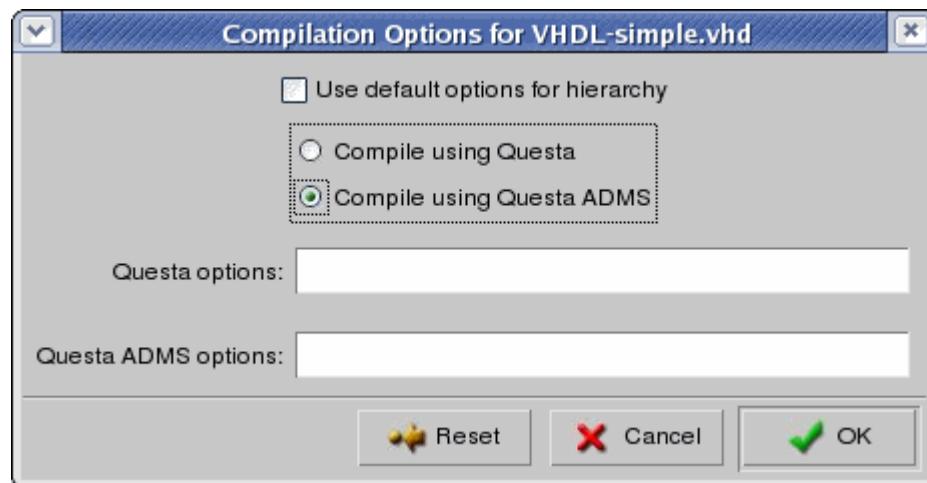


Figure 8-6. Compilation Options for <VHDL_file_name> Dialog Box



Related Topics

[Specifying Compilation Options for a Project](#)

[Importing an HDL file](#)

Compiling HDL source file(s)

Use this procedure to compile HDL source files.

Prerequisites

- The Project Navigator window is open in Pyxis Project Manager.

Procedure

1. Select the HDL file or container in the project hierarchy.
2. Run the Compile command in either of the following ways:
 - Pulldown menu — **Edit > Language > Compile**.
 - Popup menu — **Compile** menu item.
 - Project Navigator toolbar — **Compile**  menu icon.

This method may be used for an individual HDL source file or for all source files in a containment hierarchy.

Results

After the compilation is complete, a message is output indicating whether or not the compilation was successful. Select the option “Open Command Log” to view the compilation transcript.

Related Topics

[Importing an HDL file](#)

[Specifying Compilation Options for a Project](#)

[Project Navigator Toolbar](#)

Model Registration or Symbol Generation

A simulation model is considered *registered* when it has a netlist control file (NCF) and a symbol. Pyxis Netlister uses the model information in the NCF when netlisting a schematic design.

For more information on the NCF, see the [Pyxis Netlister User's and Reference Manual](#).

Pyxis Project Manager can be used to create model registrations for symbols and models contained in Pyxis Project Manager projects, external libraries, logic libraries, and technology libraries. Pyxis Project Manager supports registration of the following language types:

- EldoSPICE
- LVS
- SPICE
- System Verilog
- Verilog-A
- Verilog-AMS
- VHDL
- VHDL-AMS

Pyxis Project Manager also supports compilation of HDL files and management of pre-compiled HDL libraries. The HDL models must be compiled prior to registration. Refer to “[Language Integration for Compilation](#)” on page 453 for details.

The following features are supported by Pyxis Project Manager with respect to model registration and symbol generation:

- Generating symbols for all design units in a language file (such as a Verilog module or VHDL entity) or in a SPICE sub-circuit, and register those symbols with their source files.
- Generating a symbol automatically for an individual model or as part of a batch operation for both SPICE files and compiled HDL.
- Registering a compiled HDL model or a SPICE sub-circuit to a symbol for an individual model or as part of a batch operation.
- Allowing registration of a model from a source file within the project hierarchy.
- Selecting any compiled library from outside the project while registering a model.
- Registering a compiled design unit from an ADMS or ModelSim® library.
- Mapping the pins and properties of a symbol to corresponding values in the model.

- Specifying open pins or implicit pins.

Note

Symbols in Pyxis Project Manager can be generated using the default symbol layout or by specifying the symbol layout options through interactive registration.

Specifying the Default Registration Settings	464
Project Registration Options Dialog Box	465

Specifying the Default Registration Settings

Pyxis Project Manager supports default hierarchy settings for the following registration options: pin mapping, pin spacing, symbol shape, model ports (that may or may not be included on the generated symbol), and additional symbol pins (that may or may not be matched on the model).

Prerequisites

- The Project Navigator window is open in Pyxis Project Manager.

Procedure

1. Select the project or external library in the hierarchy for which you want to specify the default registration settings.
2. Select the pulldown menu option **Edit > Language > Registration Options**. This invokes the Project Registration Options dialog box, as shown in [Figure 8-7](#).
3. Set your default settings, which in turn are applied to all HDL source files of that type within the current project or external library.
4. Click **OK**.

Related Topics

[Model Registration or Symbol Generation](#)

[Registering a Model from a Compiled Library](#)

[Registering a Model from a Source File](#)

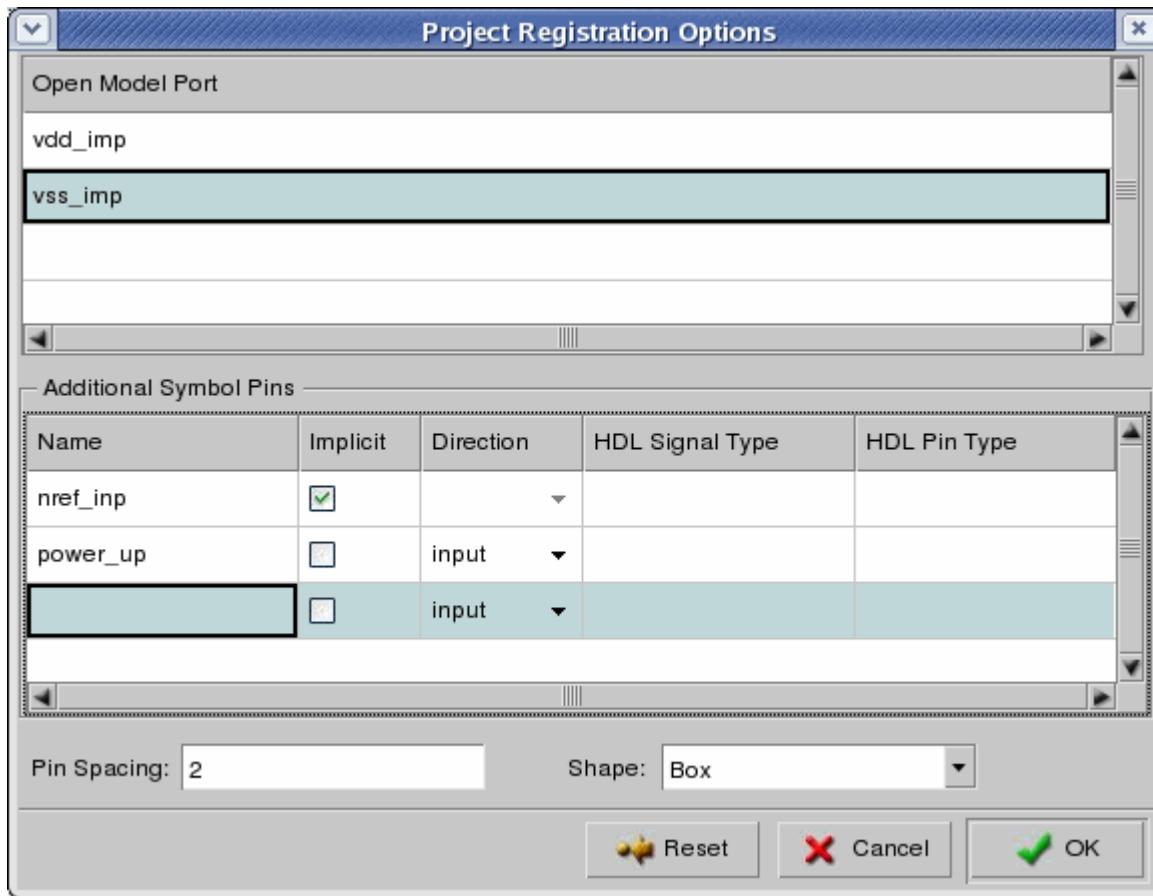
[Registering All Models from a Source File](#)

Project Registration Options Dialog Box

To access: **Edit > Language > Registration Options**.

Use this dialog box to specify registration options for pin mapping, pin spacing, symbol shape, model ports (that may or may not be included on the generated symbol), and additional symbol pins (that may or may not be matched on the model).

Figure 8-7. Project Registration Options Dialog Box



Fields

Table 8-2. Project Registration Options

Field	Description
Open Model Port	Specify the names of any ports that are not expected to match symbol pins. During registration, if these ports are found on a model, they are not added to the generated symbol, and are marked as “open” in the NCF entry for that registration.
Additional Symbol Pins	Specify any symbol pins that are not always expected to match a port on the model. During registration, if these pins are found on the symbol, and do not match any ports on the model, then they are left open. If a symbol is generated during registration, the generated symbol always includes these pins. For each symbol pin, specify the following options: <ul style="list-style-type: none">• Implicit — Indicates that a model port is not mapped to a pin but to a symbol property. If the symbol pin is implicit, then skip the remaining fields.• Direction — Specifies the direction of the symbol pin.• HDL Signal Type — Specify a value for the vhdl_signal_type property on the pin.• HDL Pin Type — Specify a value for the vhdl_pin_type property on the pin.• Pin Spacing — Specifies the space between pins on a generated symbol.• Shape — Specifies the shape of the generated symbol.

Related Topics

[Specifying the Default Registration Settings](#)

[Pyxis Netlister User’s and Reference Manual](#)

Default Symbol Layout

The default symbol layout is performed as follows:

- If the model registration targets a symbol that does not already exist, then the symbol is generated automatically.
- Symbol pins are based on the model ports for the model (from the compiled design unit data for the HDL registration, or the source file for SPICE), and the registration settings.
- Symbol properties may also be generated for any parameters for the model.

Specifying the Symbol Layout Options	467
Symbol Layout Properties Dialog Box	469
Viewing a generated symbol	471

Specifying the Symbol Layout Options

Use this procedure to override the default symbol options.

In interactive registration, you may override the default symbol options by using the Symbol Layout Properties dialog box, and specify the following symbol layout options such as: the symbol shape, the pins to be added, removed, or renamed, and the position of a pin on the symbol.

Procedure

In order to specify the symbol layout options, perform the following steps:

1. Select the symbol for which to specify layout options.
2. Begin the registration process in any of the following ways:
 - Select the pulldown menu option **Edit > Language > Register Model > Register Model**. This invokes the Register Model from Source dialog box.

OR

 - Select the pulldown menu option **Edit > Language > Register Model > From Compiled Library**. This invokes the Register Model from Library dialog box.
3. In the dialog box, make your selections for the symbol and /or library, and select a model to register.
4. If you are replacing an existing symbol, select the **Regenerate Symbol** option and click the **Symbol Layout Options** button to edit symbol pins and symbol properties.

This invokes the Symbol Layout Properties dialog box as shown in [Figure 8-8](#).
5. Make your selections in that dialog and click **OK**.

Pyxis Project Manager generates a symbol for each Verilog module or VHDL entity defined within the target source file(s) or source file(s) contained within one level of hierarchy beneath the specified container(s). The generated symbols are placed in the same component as the source file(s) in which their model is defined.

Note



After symbol generation is completed, a message displays indicating whether the operation was successful or not. To view the command log, enable the **Open Command Log** option in the [Specifying Compilation Options for a Project](#).

Related Topics

[Default Symbol Layout](#)

[Registering a Model from a Source File](#)

[Symbol Layout Properties Dialog Box](#)

[Register Model from Source Dialog Box](#)

[Register Model from Library Dialog Box](#)

[Registering All Models from a Source File](#)

[Registering a Model from a Compiled Library](#)

Symbol Layout Properties Dialog Box

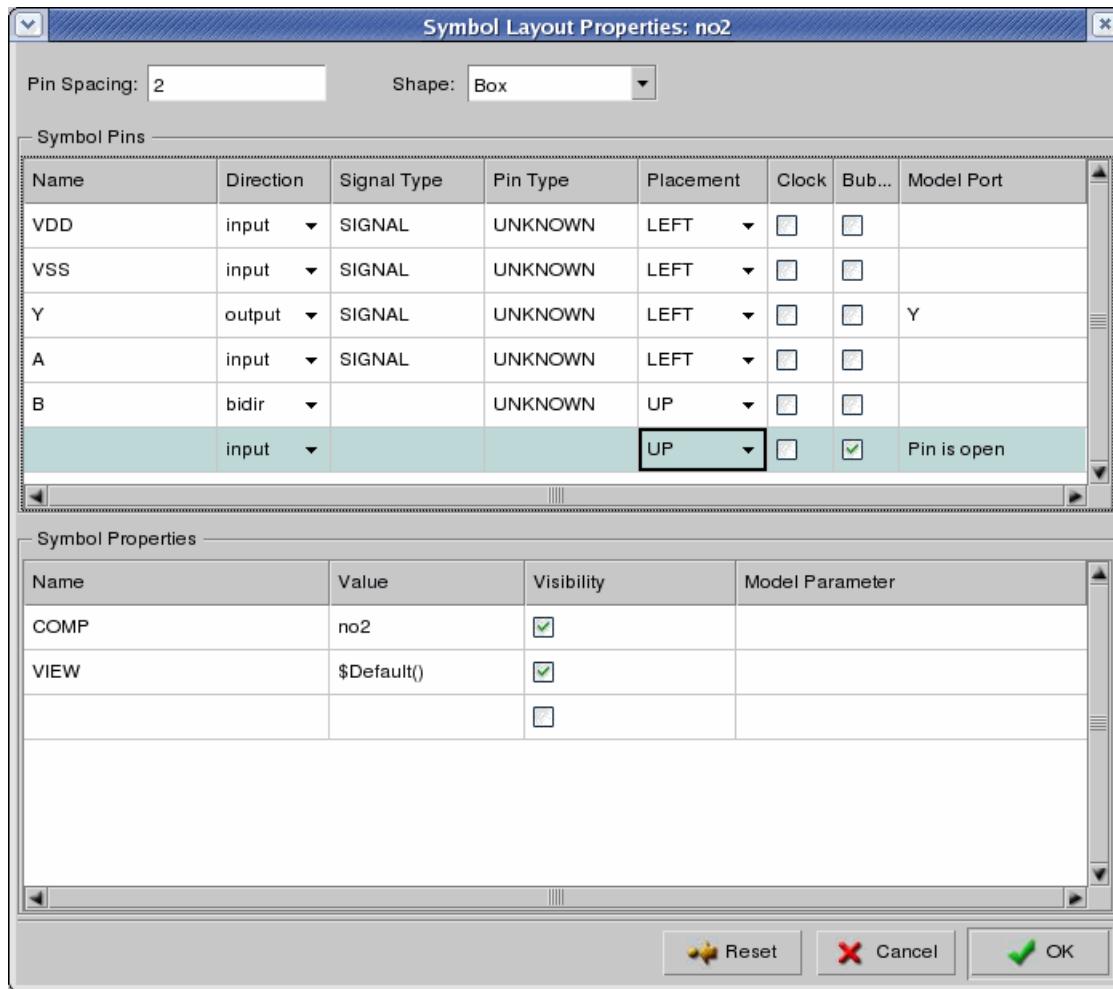
To access: Click the **Symbol Layout Options** button on the Register Model from Source dialog box or the Register Model from Library dialog box.

Generates a symbol for each model or entity.

Description

Use the dialog box shown in [Figure 8-8](#) to generate a symbol for each Verilog module or VHDL entity defined within the target source file(s) or source file(s) contained within one level of hierarchy beneath the specified container(s).

Figure 8-8. Symbol Layout Properties Dialog Box



Fields

Table 8-3. Symbol Layout Properties Dialog Box Contents

Field	Description
Pin Spacing	Specifies the space between the pins for the selected symbol.
Shape	Specifies the shape of the symbol. Options include: And Gate, Or Gate, Xor Gate, Buffer, Box, AndOr, OrAnd, Trapezoid, and Adder.
Direction	Specifies the direction of the pins. Options include: input, output, and bidir (bi-directional).
Signal Type	Specifies the vhdl_signal_type property on the pin. Options include: "signal" and "terminal".
Pin Type	Specifies the vhdl_pin_type property on the pin. Options include: "std_logic", "std_logic_vector", "electrical", and "electrical_vector".
Placement	Specifies the placement of the pins relative to the symbol. Options include: "left", "down", "up", and "right".
Clock	When selected, the generated pin is drawn to indicate an edge-sensitive input.
Bubble	When selected, the generated pin is drawn with a bubble, such as on the output of an inverter.
Model Port	Specifies the name of the model port to which this symbol pin is mapped. To edit these pin/ port mappings, use the Map Pins dialog box.
Symbol Properties > Name	Specifies the name of the symbol property.
Value	Specifies the value of the symbol property.
Visibility	Enables, disables, or customizes the visibility of the properties on the symbol model.
Model Parameter	Specifies the name of a model parameter on the HDL model that is mapped to this symbol property.

Related Topics

[Default Symbol Layout](#)

[Map Properties Dialog Box](#)

[Specifying the Symbol Layout Options](#)

[Register Model from Source Dialog Box](#)

[Register Model from Library Dialog Box](#)

[Map Pins Dialog Box](#)

[Mapping or Unmapping Symbol Pins to the Model](#)

Viewing a generated symbol

Allows you to view a generated symbol.

Procedure

- Double-click it to open it in Pyxis Schematic.
OR
- Select the **Edit Symbol** option in the Register Model from Source dialog box.
OR
- Select the **Edit Symbol** option in the Register Model from Library dialog box.

Related Topics

[Default Symbol Layout](#)

[Specifying the Symbol Layout Options](#)

[Symbol Layout Properties Dialog Box](#)

[Register Model from Source Dialog Box](#)

[Register Model from Library Dialog Box](#)

Registering a Model from a Source File

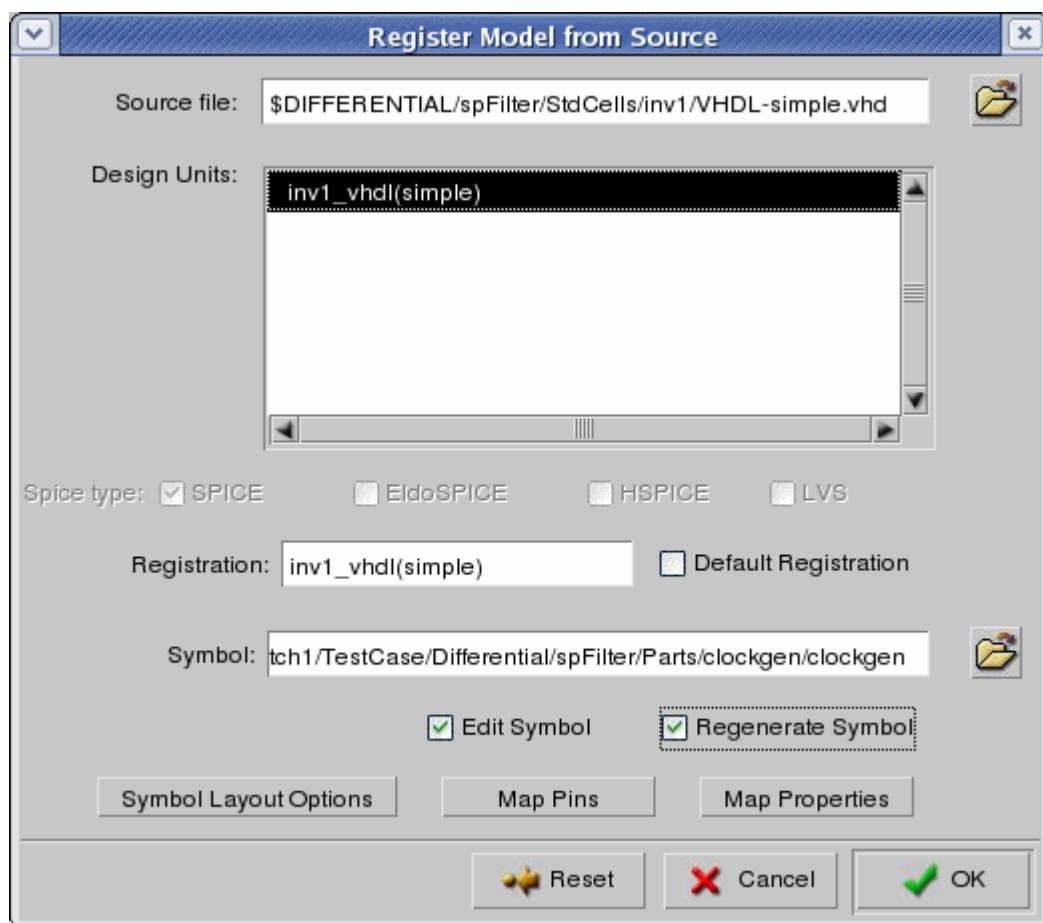
Registers the model of a specific design unit in a source file to a specific symbol.

Procedure

1. Select the symbol or source file for which you want a model to be registered.
2. Invoke the Register Model from Source dialog box, as shown in [Figure 8-9](#), in any of the following ways:
 - Popup menu — Select the **Register Model** option.
 - Pulldown menu — Select **Edit > Language > Register Model > Register Model**.
 - Project Navigator toolbar — Click on the **Register Model**  icon.

The dialog box is pre-populated with the default settings for the registration.

Figure 8-9. Register Model from Source Dialog Box



3. Type in or browse to the path of the **Source file**. The selection for this field accordingly populates the **Design Units**, **Registration**, and **Symbol** fields.

4. Select the design unit from the **Design Units** .
5. Specify a name for the registration in the **Registration** field if required.
6. Select the **Default Registration** checkbox if you require this registration to be the default one for the symbol.
7. Navigate to the symbol path by browsing or typing it in the **Symbol** field, if different from the one that is pre-populated.
8. Specify the symbol layout options by clicking the **Symbol Layout Options** button. This invokes the Symbol Layout Properties dialog box.
9. If you are replacing an existing symbol, select the **Regenerate Symbol** option.
10. Specify the pin mappings between the model and the symbol, by clicking the **Map Pins** button. This invokes the Map Pins dialog box, which displays a list of model ports.
11. Specify the property mappings between the model and the symbol, by clicking the Map Properties button. This invokes the Map Properties dialog box, which displays a list of model names.
12. If you are creating a new symbol and need to view it in the schematic editor, enable the **Edit Symbol** option.
13. Click **OK** to run the Register Model from Source dialog box.

Related Topics

Registering a Model from a Compiled Library	Registering All Models from a Source File
Specifying the Default Registration Settings	Project Navigator Toolbar
Symbol Layout Properties Dialog Box	Specifying the Symbol Layout Options
Map Pins Dialog Box	Mapping or Unmapping Symbol Pins to the Model
Map Properties Dialog Box	Mapping or Unmapping Symbol Properties to the Model

Registering All Models from a Source File

Generates symbols for and to register all models of a specific design unit in a source file.

Procedure

1. Select the source file.
2. Select the pulldown menu option **Edit > Language > Register All Models > Register All Models**.
A Registration Preview dialog box invokes enabling you to preview the models in the selected design unit before running the registration.
3. Click **OK** in the preview dialog box.

A transcript of the above task appears in the Message and Transcript Areas.

Related Topics

[Registering a Model from a Compiled Library](#) [Registering a Model from a Source File](#)

[Specifying the Default Registration Settings](#)

Registering a Model from a Compiled Library

Registers the model of a specific design unit from a compiled library to a specific symbol.

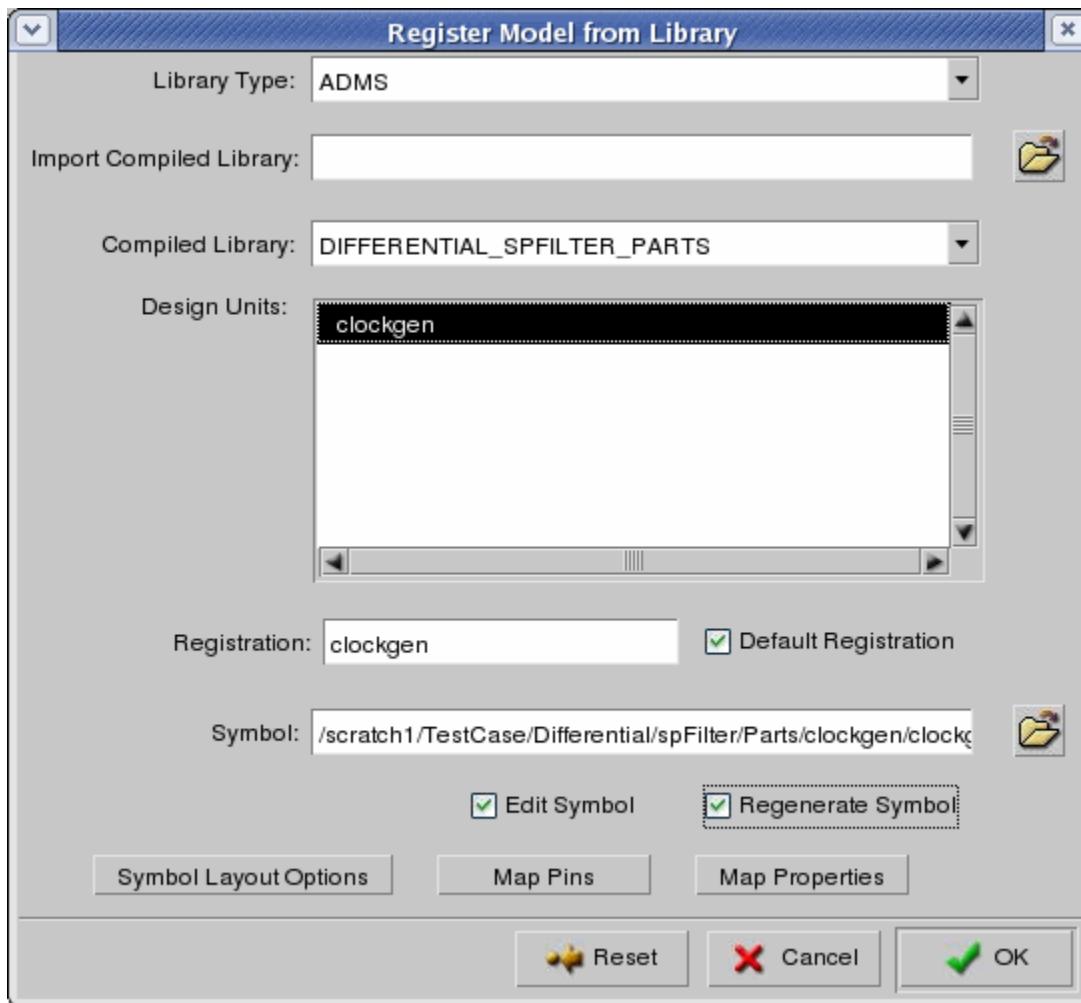
Procedure

Note

To register a model from a compiled library, the library must be included through the project's INI file. For information on INI mappings, refer to the section “[INI Mappings](#)” on page 484.

1. Select the pulldown menu option **Edit > Language > Register Model > From Compiled Library**. This invokes the Register Model from Library dialog box, as shown in [Figure 8-10](#).

Figure 8-10. Register Model from Library Dialog Box



2. Select the **Library Type** from the dropdown menu.

3. Select the **Compiled Library** from the dropdown menu.
4. Alternatively, you can browse to a compile library outside the project hierarchy by specifying it in the **Import Compiled Library** field.

The library selection accordingly populates the **Design Units**, **Registration**, and **Symbol** fields.
5. Select the required design unit from the **Design Units** list.
6. Change the name for the registration in the **Registration** field if required.
7. Select the **Default Registration** checkbox if you want this registration to be the default one.
8. Navigate to the symbol path by browsing or typing it in the **Symbol** field, if different from the one that is pre-populated.
9. Specify the symbol layout options by clicking the **Symbol Layout Options** button, which invokes the Symbol Layout Properties dialog box.
10. If you are replacing an existing symbol, select the **Regenerate Symbol** option.
11. Specify the pin mappings between the model and the symbol, by clicking the **Map Pins** button. This invokes the Map Pins dialog box, which displays a list of model ports.
12. Specify the property mappings between the model and the symbol, by clicking the **Map Properties** button. This invokes the Map Properties dialog box, which displays a list of model names.
13. If you are creating a new symbol and need to view it in the schematic editor, select the **Edit Symbol** option.
14. Click **OK** to run the Register Model from Library dialog box.

Related Topics

[Registering a Model from a Source File](#)

[Specifying the Default Registration Settings](#)

[Specifying the Symbol Layout Options](#)

[Mapping or Unmapping Symbol Pins to the Model](#)

[Mapping or Unmapping Symbol Properties to the Model](#)

[Registering All Models from a Source File](#)

[Symbol Layout Properties Dialog Box](#)

[Map Pins Dialog Box](#)

[Map Properties Dialog Box](#)

[Register Model from Library Dialog Box](#)

Mapping or Unmapping Symbol Pins to the Model

Manages symbol pin mappings during model registration.

Pyxis Project Manager provides a default pin mapping by matching pin names on the symbol with port names on the model. If the names do not match, or to override the default mapping, use the Map Pins dialog box, which can be invoked from the Register Model from Source dialog box or the Register Model from Library dialog box.

Symbol pin mappings can be managed during model registration in either the **Register Model from Library** or the **Register Model from Source** dialog boxes, as follows:

Procedure

1. Specify a symbol and model.
2. Click the **Map Pins** button in the dialog to specify the pin mappings between the model and the symbol. This invokes the Map Pins dialog box, which displays a list of model ports, as shown in [Figure 8-11](#).
3. Select the model port(s) that in turn selects the corresponding information for the model in the **Description**, **Open** and **Implicit** columns.
4. Select the symbol pin(s) that in turn selects the corresponding information for the pin in the **Description** and **Open** columns.
5. Select the **Map** or **Unmap** option to map or unmap the selected port(s) and pin(s) respectively.
6. Click **OK**.

Related Topics

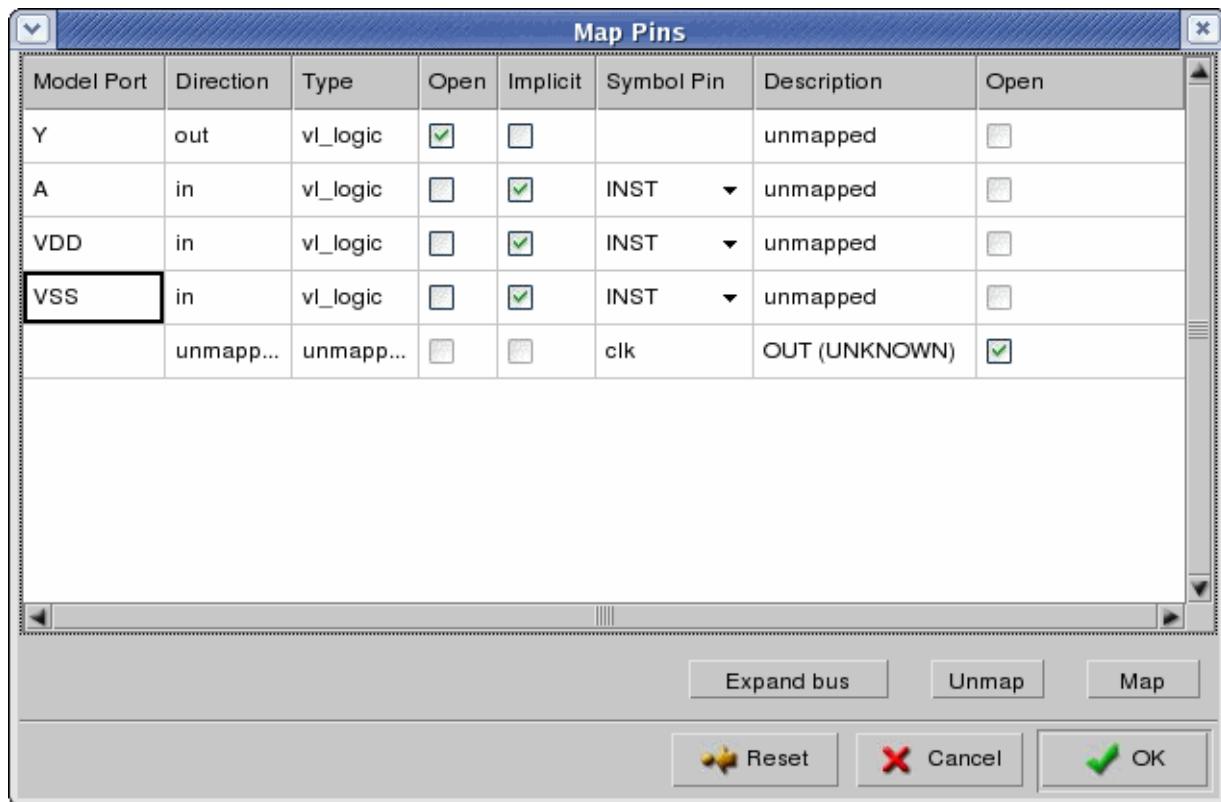
Registering a Model from a Compiled Library	Registering a Model from a Source File
Registering All Models from a Source File	Specifying the Default Registration Settings
Register Model from Source Dialog Box	Register Model from Library Dialog Box
Map Pins Dialog Box	

Map Pins Dialog Box

To access: Click the the **Map Pins** button on the Register Model from Source dialog box or the Register Model from Library dialog box.

Matches or overrides the default mapping of pin names on the symbol with port names on the model.

Figure 8-11. Map Pins Dialog Box



Fields

Table 8-4. Map Pins Dialog Box Contents

Field	Description
Model Port > Description	Specifies the direction and type of the model port.
Model Port > Open	Specifies to enable the pins to remain unmapped.
Implicit	Indicates that a model port is not mapped to a pin but to a symbol property.
Symbol Pin	When the Implicit option is checked, this column displays a pulldown list of properties associated with the pin that can be mapped to the port.
Symbol Pin > Description	Specifies the direction and type of the symbol pin.
Symbol Pin > Open	Specifies to enable the pins to remain unmapped and yet enable the registration to be valid.
Unmap	Unmaps the selected port(s) and pin(s).
Map	Maps the selected port(s) and pin(s).

Note



The registration is not valid until all the pins are matched or unless you mark them explicitly as being “open” pins.

Related Topics

- [Mapping or Unmapping Symbol Pins to the Model](#)
- [Map Properties Dialog Box](#)
- [Registering a Model from a Compiled Library](#)
- [Registering a Model from a Source File](#)
- [Registering All Models from a Source File](#)
- [Specifying the Default Registration Settings](#)
- [Register Model from Library Dialog Box](#)
- [Register Model from Source Dialog Box](#)

Mapping or Unmapping Symbol Properties to the Model

Enables you to match the symbol properties to the model or to override the default mapping.

Pyxis Project Manager provides a default mapping between model parameters and symbol properties. If the symbol properties do not match the model, or to override the default mapping, use the Map Properties dialog box, which can be invoked from either the Register Model from Library or the Register Model from Source dialog boxes.

Procedure

1. Select the symbol for which you want a model to be registered.
2. Select the pulldown menu option **Edit > Register Model > From Compiled Library**.
3. Select a model.
4. Click the **Map Properties** button to specify the property mappings between the model and the symbol. This invokes the Map Properties dialog box, which displays a list of property names, as shown in [Figure 8-12](#).
5. Select the model port name(s). This in turn selects the corresponding information for the model in the **Value** and **Type** columns.
6. Select the symbol pin name(s). This in turn selects the corresponding information for the pin in the **Value** and **Type** columns.
7. Select the **Map** or **Unmap** option to map or unmap the selected port(s) and pin(s) respectively.
8. Click **OK**.

Related Topics

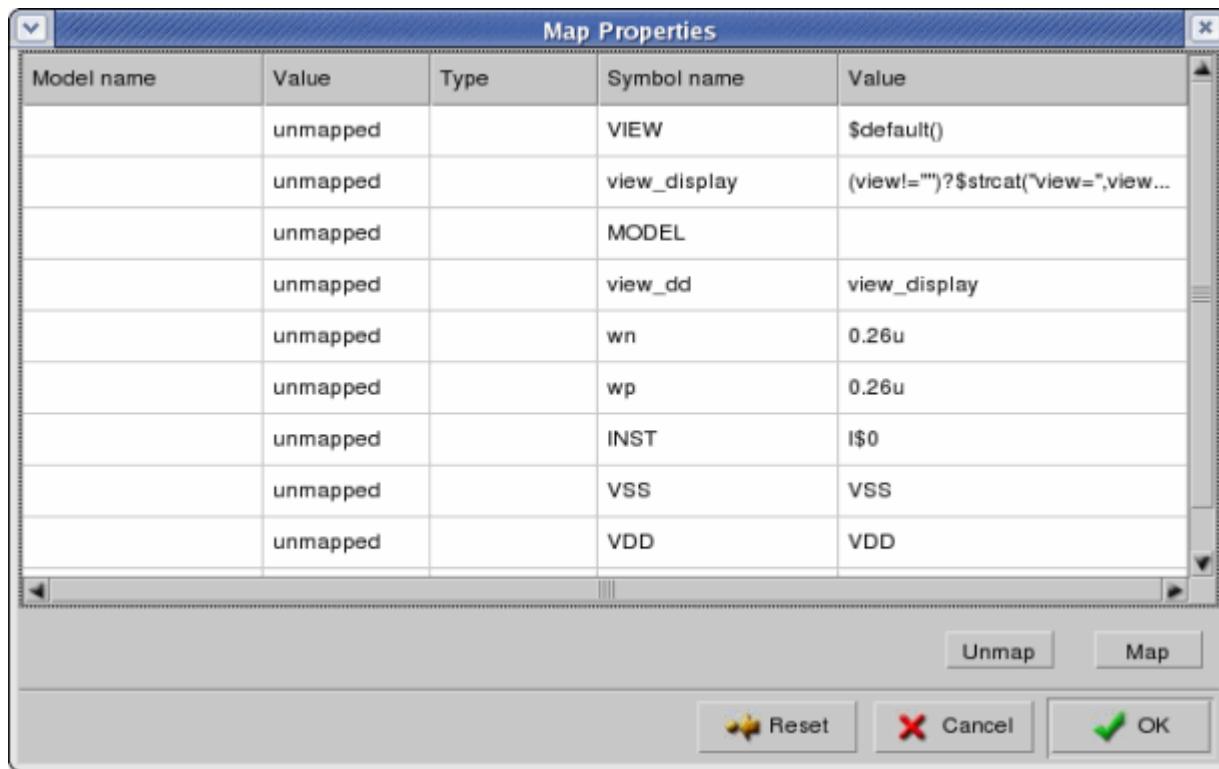
- | | |
|---|--|
| Registering a Model from a Compiled Library | Registering a Model from a Source File |
| Registering All Models from a Source File | Specifying the Default Registration Settings |
| Map Properties Dialog Box | |

Map Properties Dialog Box

To access: Click the the **Map Properties** button on the Register Model from Source dialog box or the Register Model from Library dialog box.

Matches match or overrides the default mapping of pin names on the symbol with port names on the model.

Figure 8-12. Map Properties Dialog Box



Fields

Table 8-5. Map Properties Dialog Box Contents

Field	Description
Model Name > Value	Specifies the default value of the model parameter.
Model Name > Type	Specifies the type of the parameter.
Symbol Name > Value	Specifies the value of the symbol property.
Symbol Name > Type	Specifies the type of the symbol property (such as “STRING” or “EXPRESSION”).
Unmap	Unmaps the selected port(s) and pin(s).
Map	Maps the selected port(s) and pin(s).

Note



Unlike pin mapping, the registration is valid even if some properties are unmapped.

Related Topics

[Mapping or Unmapping Symbol Pins to the Model](#)

[Map Pins Dialog Box](#)

[Registering a Model from a Compiled Library](#)

[Registering a Model from a Source File](#)

[Registering All Models from a Source File](#)

[Specifying the Default Registration Settings](#)

[Register Model from Source Dialog Box](#)

[Register Model from Library Dialog Box](#)

Checking the Language Views

Use this procedure to validate the language views.

Procedure

1. Select any container, HDL file, or symbol.
2. Select the pulldown menu option **Edit > Language > Check Language Views**.

This marks in red any container in the hierarchy that contains erroneous language files that need to be fixed.

Related Topics

[Language Import](#)

INI Mappings

Pyxis Project Manager automatically supports mapping of all compiled ADMS libraries from external libraries that are included by the current hierarchy.

Pyxis Project Manager performs automatic mapping of the external libraries in the following ways:

- Determines which external libraries are currently included in an hierarchy, either directly or indirectly, using the hierarchy's location map within Pyxis Project Manager.
- Finds the compiled libraries in each included external library.
- Compares this list of compiled libraries with the list of compiled libraries currently listed in the top level initialization (INI) file, and any differences are handled.
- Does not alter mappings to existing compiled libraries that are not in the external libraries.
- If Pyxis Project Manager determines that an additional external library has been included in the root of the hierarchy, it notifies you and automatically includes all the compiled libraries found within the external library.

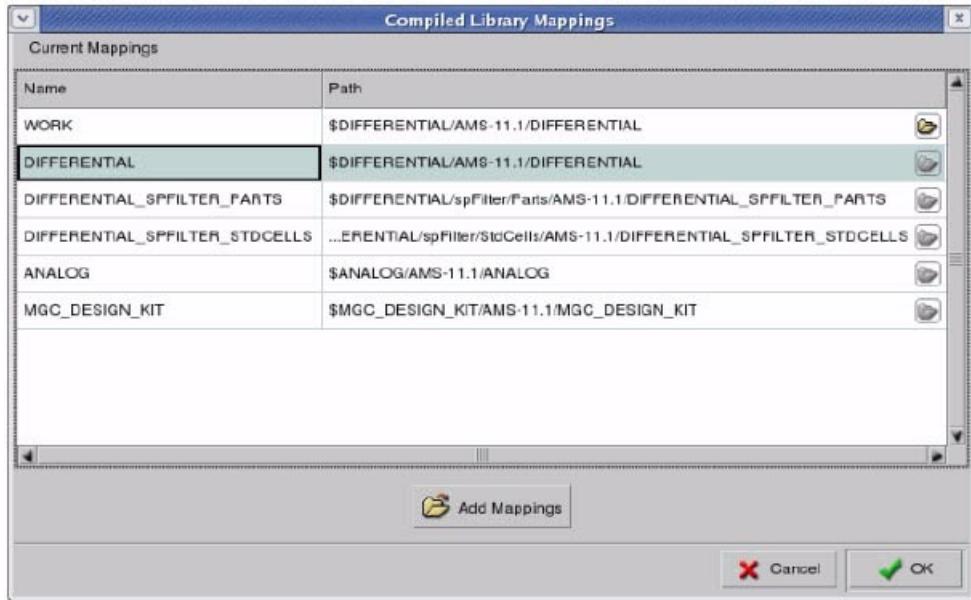
Managing INI Mappings 484

Managing INI Mappings

Use this procedure to edit the INI mappings.

Procedure

1. Select the pulldown menu option **Edit > Language > Manage INI Mappings**. This invokes the [Compiled Library Mappings Dialog Box](#) with the current mappings displayed, as shown in [Figure 8-13](#).

Figure 8-13. Compiled Library Mappings Dialog Box

2. To edit an existing mapping in the **Current Mappings** list:
 - a. Click the navigation button for the row to be edited.
 - b. In the file browser that invokes, navigate to the new path for the compiled library path, and click **OK**.
3. To delete a mapping:
 - a. Select the row in the list to delete.
 - b. Press the Delete key on your keyboard or left-click the mapping and select **Remove Entry** from the popup menu.

Note

You cannot delete entries to compiled libraries within the project hierarchy as these are managed automatically by Pyxis Project Manager.

4. To add a mapping:
 - a. Click the **Add Mappings** button.
 - b. In the file browser that invokes, navigate to the INI file or compiled library, and click **OK**. This adds the selected set of mappings to the Current Mappings table.
5. Click **OK** in the Compiled Library Mappings dialog to save the selections.

Note



The **Edit > Language > Manage INI Mappings** option may be used to add or remove mappings to compiled libraries that are not in a Pyxis Project Manager hierarchy. Compiled libraries that are contained in the project or in an external library are managed automatically when that project or external library is opened in Pyxis Project Manager.

Related Topics

[INI Mappings](#)

Language Import

Importing an HDL or a SPICE file is performed by copying the file, compiling if applicable, and optionally generating symbols for some or all of their design units.

For importing HDL, since the HDL file is compiled, you can specify compilation settings. The full model registration interface is not available for import, but you can specify any model ports that should not map to symbol pins (or vice-versa).

If a mismatch exists between the model ports and symbol pins, then the “Map Pins: <design_unit_name>” dialog box invokes automatically enabling you to fix the mismatched pin names and keep continuing with the language import process.

Importing an HDL file	487
Importing a SPICE file	490

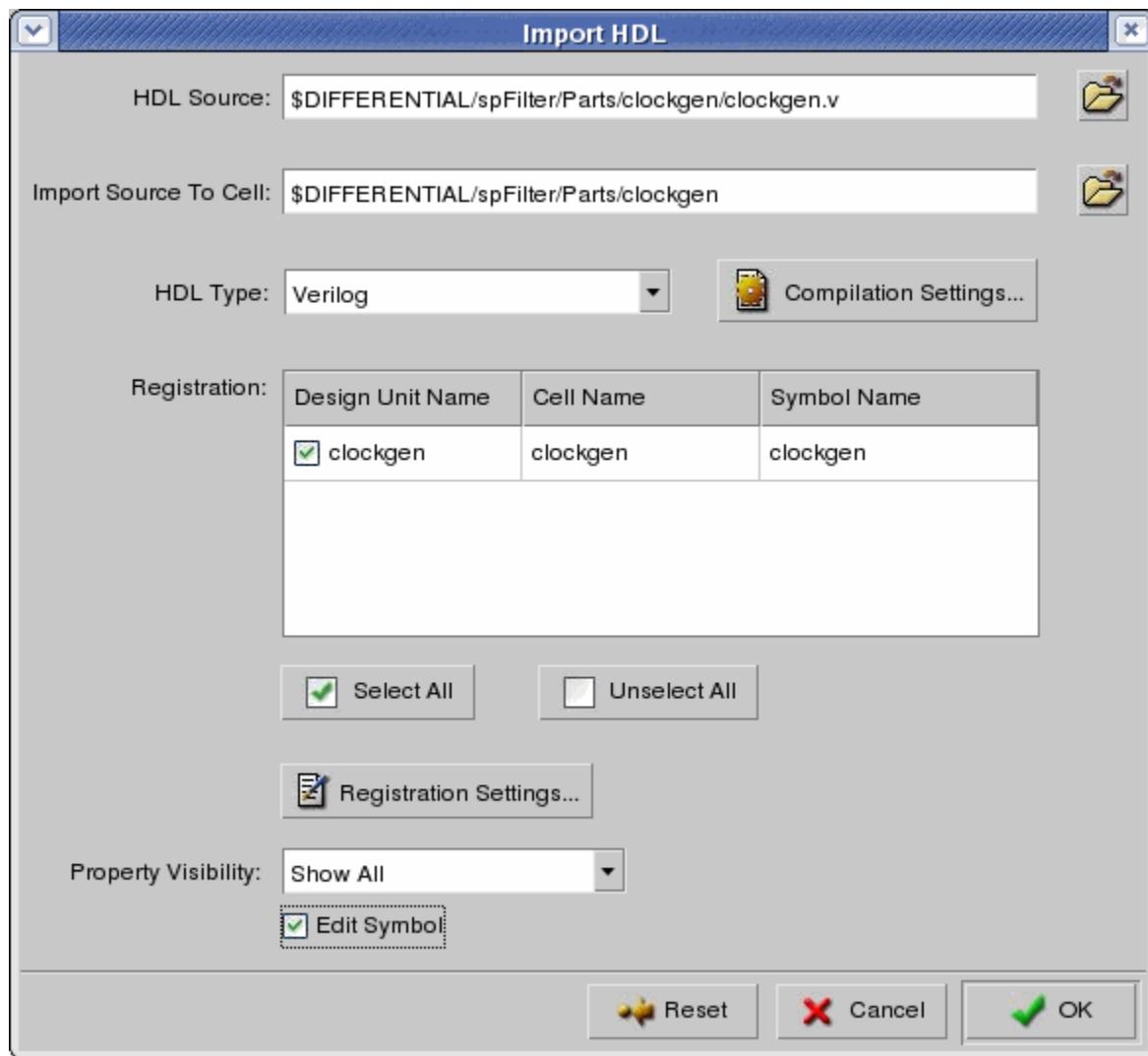
Importing an HDL file

To import an HDL file, perform the following steps:

Procedure

1. Select the pulldown menu option **File > Import > HDL**. This invokes the Import HDL dialog box, as shown in [Figure 8-14](#).

Figure 8-14. Import HDL Dialog Box



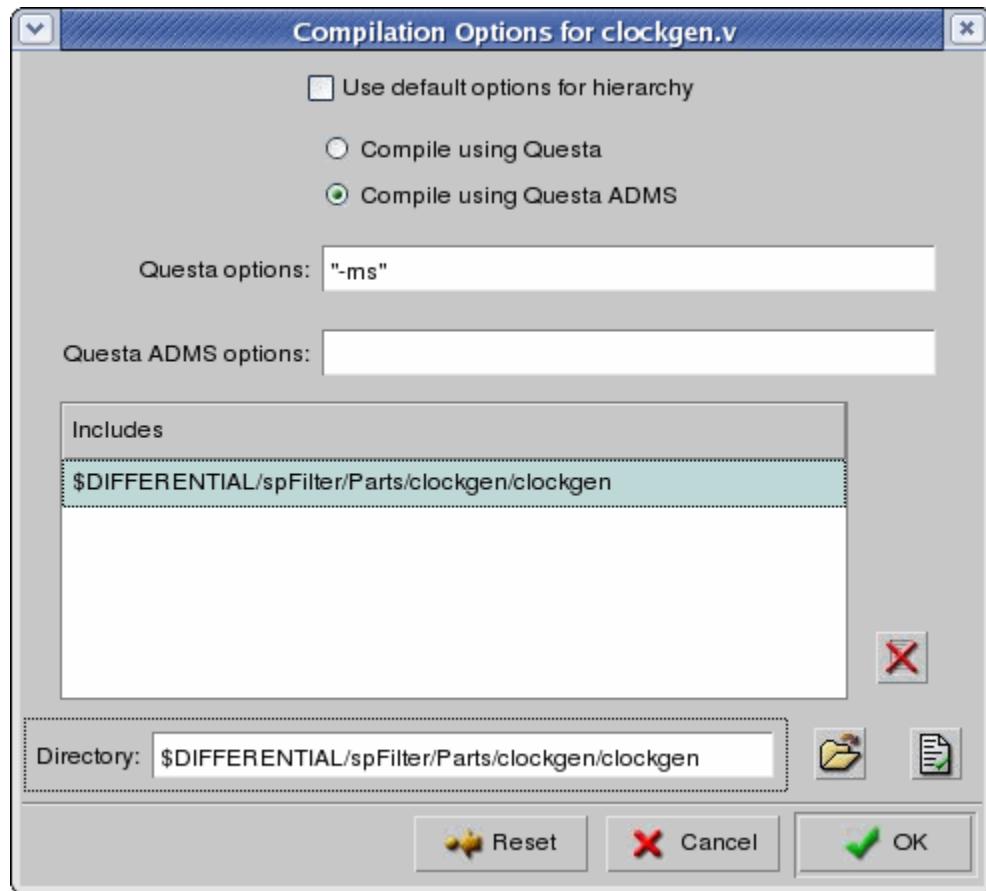
2. Navigate to the **HDL Source** file to select the design units that should be registered to symbols.
3. Select the destination cell for the imported HDL file in **Import Source to Cell**. The **HDL Type** is deduced from the HDL file extension.
4. Select the design units to import in the **Registration** table and specify the destination cell name as well as the name of the symbol that is created for them (if these columns are not already populated).
5. Select the **Edit Symbol** option if there is only exactly one design unit to import. When the new symbol is created for the imported design unit, it is opened for editing in Pyxis Schematic.
6. As the HDL file is compiled, you can specify the compilation settings as follows:

- a. Click on the **Compilation Settings** button. This invokes the Compilation Options for <HDL_Source_Filename> dialog box, as shown in [Figure 8-15](#).
- b. Select the **Use default options for hierarchy** checkbox option, or select your own settings for compilation.
- c. Click **OK** to return to the parent dialog box.

Note

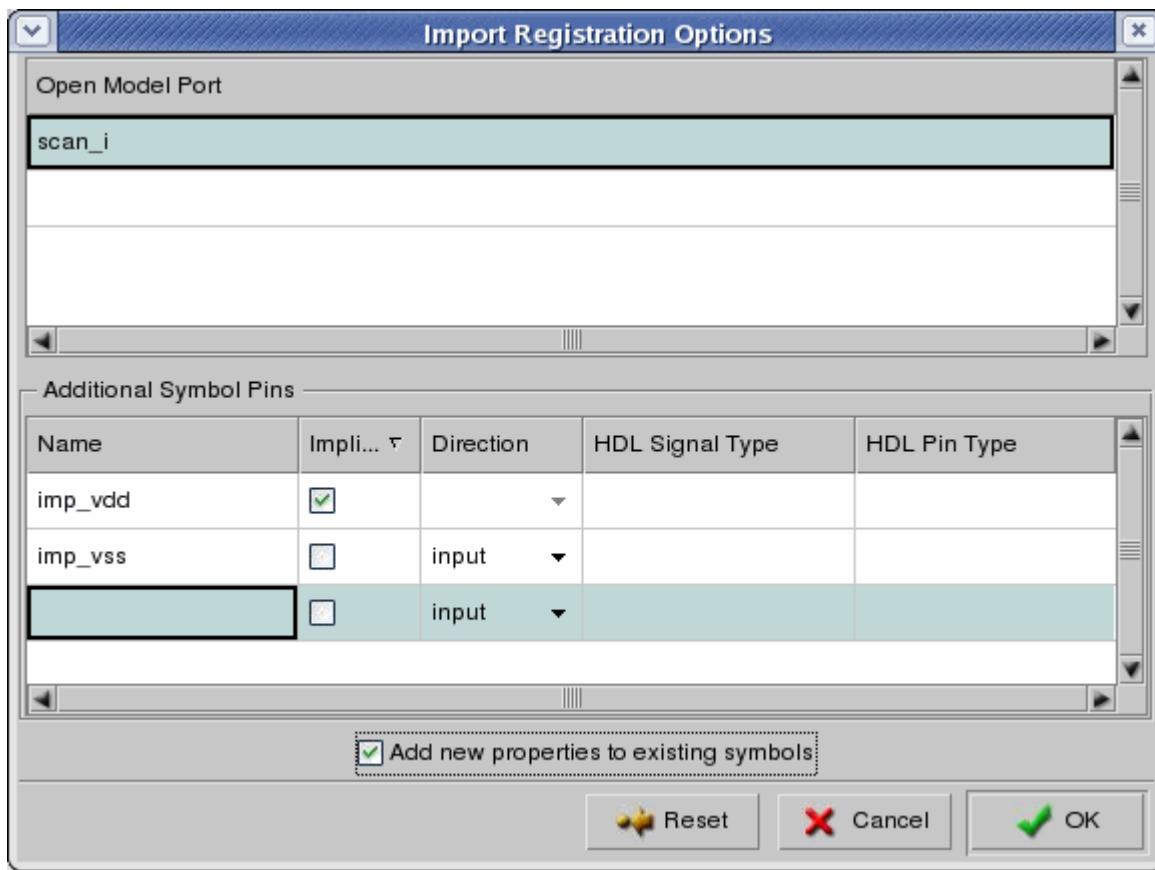
For HDL imports, symbols are generated for the selected design units.

Figure 8-15. Compilation Options Dialog Box



7. Specify the registration settings as follows:
 - a. Click on the **Registration Settings** button. This invokes the Import Registration Options dialog box, as shown in [Figure 8-16](#).

Figure 8-16. Import Registration Options Dialog Box



- b. Select your settings for the model port and symbol pins.
- c. Click **OK** to return to the parent dialog box.
8. Click **OK** to run the Import HDL dialog box with the specified settings.

Related Topics

[Language Import](#)

[Import HDL Dialog Box](#)

[Importing a SPICE file](#)

[Import Registration Options Dialog Box](#)

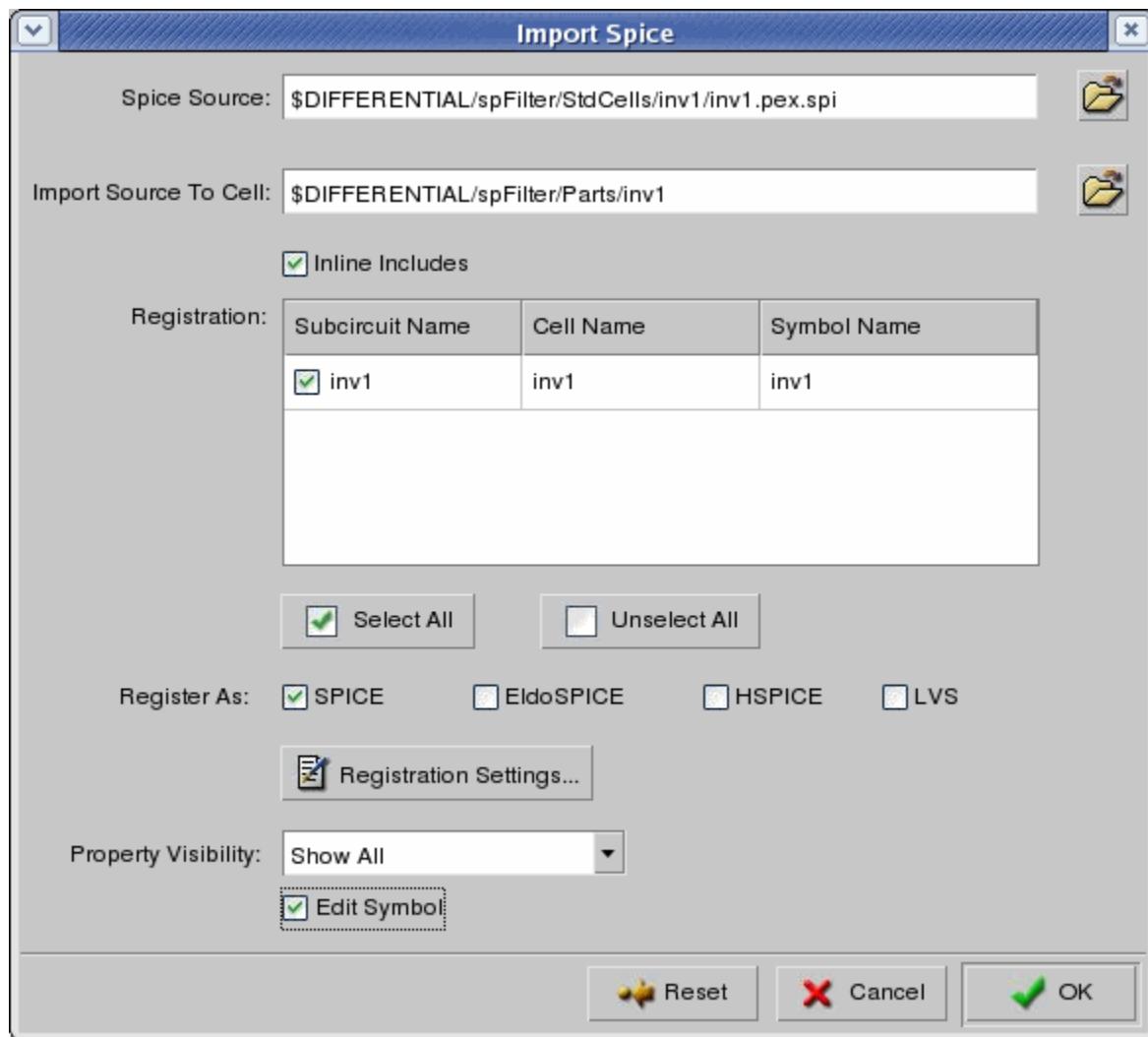
Importing a SPICE file

Importing a SPICE file is performed by copying the file and optionally generating symbols for some or all of its sub-circuits.

Procedure

1. Select the pulldown menu option **File > Import > Spice**. This invokes the Import Spice dialog box, as shown in [Figure 8-17](#).

Figure 8-17. Import Spice Dialog Box



2. Navigate to the **SPICE Source** file for which to select any SPICE sub-circuits that should be registered to symbols.

Note

For SPICE imports, symbols are generated for the selected sub-circuits.

3. Select the destination cell for the imported SPICE file in **Import Source to Cell**.
4. Select the subcircuits to import in the **Registration** table and specify the destination cell name as well as the name of the symbol that is created for them (if these columns are not already populated).
5. Select one or more options in **Register As** based on how you intend to use the models for netlisting and/or simulation.

6. Select the **Edit Symbol** option if there is only exactly one design unit to import. When the new symbol is created for the imported design unit, it is opened for editing in Pyxis Schematic.
7. Specify the registration settings as follows:
 - a. Click on the **Registration Settings** button. This invokes the Import Registration Options dialog box, as shown in [Figure 8-16](#).
 - b. Select your settings for the model port and symbol pins.
 - c. Click **OK** to return to the parent dialog box.
8. Click **OK** to run the Import Spice dialog box with the specified settings.

Related Topics

[Language Import](#)

[Import Spice Dialog Box](#)

[Importing an HDL file](#)

[Import Registration Options Dialog Box](#)

Appendix A

Troubleshooting

View troubleshooting information and procedures that you can use when a Pyxis Project Manager problem occurs.

Salvage a Damaged Design Object.....	493
Problems with Type Registries	503

Salvage a Damaged Design Object

Learn when and how to salvage a design object and how to avoid potential problems during salvaging. These procedures use nearly all of the Pyxis Project Manager shell commands.

 **Note** _____
If the following error message is generated, an exception has occurred and you cannot use the procedures in the appendix:

Unrecoverable error. Attempt to save your data and exit the application.

In this case, you should attempt to save your data and exit the Pyxis Project Manager. If attempting to save your data causes another exception to be generated, you must exit the Pyxis Project Manager with a loss of data.

If a design object is damaged by an application failure, some temporary files may be left behind in its fileset. When this failure happens, the existing versions of your design object remain intact, unless a disk hardware failure damaged the file system itself. However, you cannot edit the design object, and you may not be able to read it. In addition, many Pyxis Project Manager commands may no longer work on the object. As a result, you must use the **salvage_object** shell command to restore it to a usable state. If you understand how salvaging works, you may proceed directly to the salvaging procedure described in “[The Salvage Process](#)” in this appendix.

 **Note** _____
For information on the shell commands used in these procedures, refer to “[Shell Command Dictionary](#)” in the *Pyxis Project Manager Reference Manual*.

To understand how salvaging works, you first need to know something about the temporary lock, edit, and new edit files. Before reading these procedures further, please read “[Design Object Filesets](#),” in Appendix C.

Note

In addition to the **salvage_object** shell command, the Pyxis Project Manager also provides **Object > Salvage Object** from the navigator's pulldown menu.

Finding a Design Object's Name and Type	494
Recognizing When a Design Object Is Damaged.	495
The Salvage Process	496
Salvaging by Deleting Edit Files	498
Salvaging with Edit Recovery	498
Protecting Your Data During a Salvage	499

Finding a Design Object's Name and Type

In general, to salvage a damaged design object, you must specify its name and its type to the **salvage_object** command.

Procedure

1. Use **cd** to change your working directory to the location of the design object that you want to copy.
2. Execute the following command:

```
$ list_contents .
```

The design objects that reside in the current working directory are displayed, one per line, in the following format:

```
./my_object:my_type
```

In this format example, *my_object* is the name of the design object, and *my_type* is its type.

If one of the following errors occurs, see your system administrator:

- The **list_contents** command cannot recognize your design object. This happens if the type registry that contains your type is not loaded into your environment. In this case, **list_contents** lists the individual members of your design object's fileset as design objects. These files look like this:

```
./my_object:Mgc_container
./my_object.my_type.attr:Mgc_file
./my_object.suffix1:Mgc_file
./my_object.suffix2:Mgc_file
...
...
```

In this example, the fileset of *my_object* consists of a directory and several files, each with a unique suffix. Notice the file with the “.attr” suffix. That is the attribute file for the design object; the object's type, *my_type*, is embedded into its name.

- The operating system cannot find **list_contents** or any other design management shell command. Note that all design management commands reside in **\$MGC_HOME/bin**.

Related Topics

[Salvage a Damaged Design Object](#)

Recognizing When a Design Object Is Damaged

If you view the fileset of a damaged design object, you can see the temporary files whose presence indicates a damaged object.

However, these files could also be present during any valid edit session. To determine that a design object is damaged, you need more than only the presence of the temporary files.

If you have already invoked an application on a design object, and the application returned a message that the design object has a malformed lock, that object is damaged, and you must salvage it. A lock is malformed if the editing process that owns it is no longer running. In this case, the process probably died during a system failure.

Procedure

If you do not know whether or not a design object is damaged, you should look at the fileset of the design object, by performing the following steps:

1. List the files of any design objects named *my_dobj*, by executing the following command:

```
$ /bin/ls my_dobj*
```

2. Look for a lock (*.lck*) file, and any edit (*.ed*) or new edit (*.ed.new*) files.

If the design object has neither, it is not damaged and does not need salvaging. If you see the lock file in the output of this command, some application is using the design object, or the object is damaged.

3. If you think that no application is using the design object, verify this assumption by trying to invoke an application on it, or by running the following **change_references** shell command:

```
$ change_references qqq qqq my_dobj
```

Because the matching and replacement patterns are the same, this command does not actually make changes, even in a usable design object. Its only purpose is to check the

integrity of the lock. If the lock is valid, someone else is actually using the design object, and the command tells you that the lock belongs to another lock manager. If the lock is malformed, the command tells you so, and you know that the design object is damaged.

If edit (.ed) or new edit (.ed.new) files are in the fileset, but no lock file is present, the design object is not in a normal state. It is usable, and you can read any existing version, or can edit the current version. However, the edit files must be left over from some earlier edit session whose lock is now gone. If you edit the design object in this state, your new edits overwrite the leftover edit files.

If you want the edits that are contained in these edit files, you should not edit the design object further. Instead, you should do the following:

- Treat the object as though it is damaged, and try to recover the edits by using the **salvage_object** command.
- To ensure that no one else edits the object, restrict write access to the object until you have salvaged it.

Related Topics

[Salvage a Damaged Design Object](#)

The Salvage Process

Every salvage operation has two main parts:

- Removing the malformed lock (if it exists)
- Removing or incorporating all or some of the edits that were made before the application failure

Remove the Malformed Lock

The lock is kept in the temporary lock file. To repair a design object with a malformed lock, remove the lock file by using the **salvage_object** command, which can recognize whether or not a lock is malformed. If the lock is malformed, the command removes the lock file for you. Removing a malformed lock makes the design object readable and editable again.

If you remove a lock file by using a UNIX operating system command, you risk allowing any user with write privileges on the design object to edit it further, thus destroying any edit files that already exist. You also risk removing a valid lock.

You cannot remove a lock if you do not have delete privileges on the design object. The design object remains damaged as long as the lock file exists, preventing you from doing most normal operations. To proceed with your work, you must either get the necessary privileges to remove the lock or see your system administrator.

Recover Edits

The *.ed* and *.ed.new* files contain data for your design object that your application wrote to disk during the edit session that failed. If you remove the malformed lock and start a new edit session, your application reads the current version of the design object, not the edit files. If you write any data to disk, the application overwrites the existing edit files. When used with no options, the **salvage_object** command simply deletes all existing edit and new edit files. Either way, you lose the data from the failed edit session.

Instead, you may want to keep the failed session's edits. To try to recover the edits, do the following:

- Use the **-r** option with the **salvage_object** command.

With this option, the command attempts to recover the edits by writing them into a new version of the design object. The command applies version pruning to versioned design objects. The command simply overwrites the current design data of unversioned objects.

Reasons for the edit recovery to fail include the following:

- Incomplete set of *.ed* files. If *.ed* files are not present for each design file in the fileset, **salvage_object** cannot recover any edits. If the design object has an attribute file, the complete set includes the *.attr.ed* file.
- Inconsistent fileset. If the *.attr.ed* file is present, **salvage_object** reads it and checks for inconsistency in the entire fileset. For example, if another user edited your design object's fileset using operating system commands, that user might have corrupted the fileset by deleting or misnaming a file. The **salvage_object** command can often detect this.

If recovery fails, you know that the command found a definitive reason why the *.ed* files cannot be used to write a new version of the design object. For this reason, the *.ed* files are of no use, so **salvage_object** deletes them. You can use the salvaged design object, but your edits from the session that failed are gone.

Although the recovery may succeed, you are not guaranteed to have a useful new version. After any salvage operation, you must examine the result very carefully to determine if the current version contains flaws that the **salvage_object** command could not find. If you miss a flaw now, it may persist into future versions. If you find a flaw, you can either correct it in the next edit session, or you can use the **revert_version** command to go back to the previous version, thus throwing away the failed session's edits.

The *.ed.new* files are not useful in salvaging a design object. Although they contain data that was written to disk, they are simply temporary files that help in the process of writing edits to disk. As a result, they are an incomplete version of the fileset, and **salvage_object** always deletes them.

Related Topics

[Protecting Your Data During a Salvage](#)

Salvaging by Deleting Edit Files

If you do not want the edits of a failed edit session, you have the easiest kind of salvaging job.

Procedure

To salvage design object *my_obj* of type *type1* and discard existing *.ed* files, execute the following command:

```
$ salvage_object my_obj:type1
```

If the design object has a malformed lock, the *.lck* and all *.ed* and *.ed.new* files are deleted.

Though you lose the edits from the failed application session, this operation involves no risk to the design object as a whole, and the salvage operation is very quick and easy.

Related Topics

[Salvage a Damaged Design Object](#)

Salvaging with Edit Recovery

Salvaging with edit recovery is only appropriate in restricted circumstances. It can be safely used only when all of the following conditions apply:

- Your design object has an attribute file.
- Your design object is versioned.
- Your design object's version depth is at least 2.
- You do not care if your oldest existing version is pruned.

If you use this procedure under conditions other than those stated here, you risk losing data. While these conditions appear restrictive, they are common.

Procedure

To salvage design object *my_obj* of type *type1* and recover existing edits (if possible), do the following:

- Use **salvage_object** with the **-r** (recover) option, as shown:

```
$ salvage_object -r my_obj:type1
```

If the design object has a malformed lock, the lock file is deleted. If the edits can be recovered, the edits in the *.ed* files become the current version of the design object. For versioned design objects, any versions beyond the version depth are pruned. If the edits cannot be recovered, the *.ed* files are deleted.

Caution



Remember that if recovery succeeds, you should always check the new version thoroughly for correctness. Success of the recovery does not necessarily mean that the new version is error-free. It only means that the **salvage_object** command could not find any problems.

Related Topics

[Salvage a Damaged Design Object](#)

Protecting Your Data During a Salvage

In certain cases, the previous easy method of recovering edits has some risks.

This process is designed to minimize any dangers inherent in salvaging edits. You can use this procedure any time that you want to be very careful about salvaging edits. You should use it if any of the following conditions is true:

- Your design object has no attribute file.
- Your design object is not versioned.
- Your design object's version depth is 1.
- You want to keep your oldest existing version safe from version pruning.

The fundamental idea of this procedure is to preserve all of the existing versions of the damaged design object throughout the salvage operation. Version preservation is very important if, for any reason, only one version of the damaged design object exists. If only one version of the damaged design object exists, and if edit recovery succeeds, the recovered edits form the new and only version of your design object. If you then check that version and find that it still has flaws that cannot be readily corrected, you cannot revert to an earlier version because that version has already been deleted. In this case, you can only recover your previous version by restoring the design object from a backup tape.

Before salvaging, you can protect previous (or the only) versions of the damaged design object in one of three ways:

- Freeze the version that you want to preserve, by using the **freeze_version** command.

You can do this only if your design object has an attribute file and is versioned. After successfully salvaging, you must remember to unfreeze the version, if you no longer want it.

- Increase the version depth, by using the **set_version_depth** command.

You can only increase the version depth if your design object has an attribute file and is versioned. If you do not know the current version depth, you should set it to any depth that is at least one greater than the number of versions that the design object currently has. By setting it to a depth at least one more than the current number of versions, you ensure that the oldest version of the design object is not pruned during a successful recovery.

- Copy the damaged design object, by using the **copy_object** command.

You can always copy the damaged design object. This is the safest of the three methods because you can always return to the starting point of the salvage operation. However, it uses more disk space than options 1 and 2.

Another procedure for salvaging with edit recovery, and the *safest*, requires ten steps and uses both the project manager shell commands and standard operating system commands.

Procedure

To safely salvage design object *broken_obj* of type *type1* with edit recovery, perform the following steps:

1. Verify that the design object needs salvaging.
 - a. View the design object's fileset. To list the files of all design objects named *my_dobj*, execute the following command:

```
$ /bin/ls my_dobj*
```
 - b. In the output of the **ls** command, look for a lock (*.lck*) file, and any edit (*.ed*) or new edit (*.ed.new*) files.
 - o If the design object has neither lock files nor edit files, it does not need salvaging. You can stop this procedure.
 - o If it has a lock file, proceed to the step 2.
 - o If it has edit files but no lock file, proceed to step 3.
2. If the design object has a lock file, ensure that the lock is malformed before you begin the salvage operation to avoid removing another user's valid lock. To test the design object's lock, use the **change_references** shell command, as shown:

```
$ change_references qqq qqq my_dobj
```

Because the matching and replacement patterns are the same ("qqq"), this command does not actually make changes, even if it succeeds. Its only purpose is to check the

integrity of the lock file. Because the design object has a lock file, this command is guaranteed to fail and provides the following information:

- a. If the lock is valid, someone else is actually using the design object, and the command fails and tells you that the lock belongs to another lock manager. In this case, your design object does not need salvaging, and you should stop this procedure.
- b. If the lock is malformed, the command fails for that reason and tells you so. In this case, proceed to the step 3.
3. Ensure that you have write and delete privileges for the design object, as given by the file system. Also, remove these rights for other users of your computer system.

Caution



Be careful that as you remove privileges for other users, you do not remove your own privileges. If you lack write and delete privileges, the **salvage_object** command fails, and you lose the edit files that you are trying to recover.

You must ensure these things because during this procedure, you remove the malformed lock before recovering edits. Removing the lock opens the design object to further editing by other users, which would defeat the purpose of your salvage operation.

4. To ensure that the status of the design object has not changed since you began this procedure, repeat step 1.
5. If the design object has edit files but no lock file, proceed to step 6. If it has a lock file, remove the lock by executing the following command:

```
$ /bin/rm broken_obj.type1.lck
```

Because you have protected previous versions of your data, and because you have restricted write access to only yourself, removing the lock file this way does not jeopardize your data.

6. Protect the design object from the dangers of salvaging, using one of the three methods described previously. Perform one, and only one, of the following:
 - To set the version depth to protect the oldest version, assuming that the damaged design object has two versions, execute the following command:

```
$ set_version_depth -d 3 broken_obj:type1
```
 - To freeze the oldest version, assuming that it is number 6, execute the following command:

```
$ freeze_version broken_obj:type1[6]
```
 - To copy the damaged design object, execute the following command:

```
$ copy_object broken_obj:type1 backup_obj
```

7. Salvage the original design object, specifying the **-r** (recover) option, by executing the following command:

```
$ salvage_object -r broken_obj:type1
```

Caution



Remember that if recovery succeeds, always check the new version thoroughly for correctness. Success of the recovery does not necessarily mean that the new version is error-free. It only means that the **salvage_object** shell command could not find any problems.

If the salvage is not successful, proceed to step 9.

8. Check your salvaged design object.

If the salvage is successful, check the design object thoroughly by invoking your application on it. If the new version looks correct, proceed to step 9. If the new version has un-correctable errors, the edits that you wrote into the new version are useless and you should discard them and do the following:

- a. If your design object is versioned, revert to the previous version by executing the following command:

```
$ revert_version broken_obj:type1
```

- b. If your design object is not versioned or has no attribute file, destroy it, replace it with the backup copy, and salvage again without edit recovery. To perform these actions, execute the following commands:

```
$ delete_object broken_obj:type1
$ move_object backup_obj:type1 broken_obj
$ salvage_object broken_obj:type1
```

9. Clean up the temporary effects of salvaging.

Depending upon which method that you used in step 6 to protect your data before salvaging, you may need to reset the version depth, unfreeze a frozen version, or delete the backup design object.

10. Reset the file system privileges to allow other users access to the salvaged design object.

Related Topics

[Salvage a Damaged Design Object](#)

Problems with Type Registries

If the type of a design object on which you want to work is missing from the Pyxis Project Manager type manager, the Pyxis Project Manager cannot recognize the object. As a result, Pyxis Project Manager commands do not work properly on that object.

The following text describes how to determine that a design object type is missing and how to add a missing type to your environment.

Determining That a Type is Missing	503
Adding a Type to Your Environment	504
Adding a Type through the Type Window	505
Adding a Type with the MGC_TYPE_REGISTRY Shell Variable	506

Determining That a Type is Missing

If a design object type is missing from the Pyxis Project Manager type manager, when you navigate to that design object, the Pyxis Project Manager does not display its name and distinctive icon.

Instead, the design object's fileset appears as a collection of simple files and directories. These files have generic file and directory icons, and their names appear as follows:

```
./my_object
./my_object.my_type.attr
./my_object.suffix1
./my_object.suffix2
...
...
```

In this example, the fileset of *my_object* consists of a directory and several files, each with a unique suffix. Notice the file with the “.attr” suffix. This file is the attribute file for the design object, and the object's type *my_type* is embedded in its name. In this example, *my_type* is missing from the Pyxis Project Manager type manager.

You can use the `$check_registries()` function to error check your type registry directories when the Pyxis Project Manager cannot recognize design objects or icons. This function invokes the script `$MGC_HOME/bin/check_rgy`, which compares the directories found in `$MGC_HOME/registry` with the directories found in `$MGC_HOME/shared/registry`. This function checks the subdirectories `type_registry`, `font_registry`, `fonts`, and `tcodes`.

Procedure

1. Choose **Setup > Check Registries** from the Pyxis Project Manager session window.
2. Select the node option for shared MGC tree from the current node, or some other node.

3. If you specified some other node as the option, then enter the Node name at the prompt.
4. Click OK to execute the Check Registries dialog box.

A read-only window displays the output. If the sub-directories are current, the report displays “OK”. If other messages appear, such as “DOES NOT MATCH” or “MISSING”, contact your System Administrator.

Related Topics

[Problems with Type Registries](#)

[\\$check_registries\(\)](#)

Adding a Type to Your Environment

In the Mentor Graphics Tree, design object type descriptions are stored in special files called *type registries*.

In general, Mentor Graphics software packages name their type registry file the following:

`$MGC_HOME/shared/pkgs/pkgname_rgy/registry/type_registry/pkgname.rgy`

In addition, you may have your own type registries defined, which can reside anywhere.

When you invoke the Pyxis Project Manager, the types that are contained in your Mentor Graphics Tree are loaded into the Pyxis Project Manager *type manager*. As a result, these types are available to the Pyxis Project Manager. If you want to add a new design object type to the type manager, or if you determine that a type that you expected to be in the type manager is missing, you need to add one or more type registries to the type manager.

Procedure

1. In the Pyxis Project Manager type window, execute the `$load_registry()` function.
2. In the shell, prior to invoking the Pyxis Project Manager, set the `MGC_TYPE_REGISTRY` environment variable.

The `MGC_TYPE_REGISTRY` variable is like any other shell environment variable; you can set it and echo its contents. When used in the shell from which you invoke your Pyxis Project Manager, it specifies any number of type registries to be added to the current environment's type manager.

Related Topics

[Problems with Type Registries](#)

Adding a Type through the Type Window

Add a design object type through the type window.

Procedure

1. With the session window active, enter the following in a popup command line:

```
open types window
```

A type window appears, listing the types that are known by the Pyxis Project Manager type manager.

2. Choose **Load Registry** from the type window's popup menu, which displays a prompt bar.
3. Enter the pathname of the type registry in the Registry Path field and execute the prompt bar.

The type window updates its display to show the types contained in the type registry that you added.

As a result of your loading the registry, the Pyxis Project Manager knows about your tool and data objects, and can recognize and manipulate them.

When you add a type while in the Pyxis Project Manager, using the Load Registry command, existing navigators do not update to recognize previously unrecognized design objects. For these navigators to recognize design objects of the newly added types, execute **Update Window** from the navigator's popup menu.

If you want a particular type registry to automatically load each time that the Pyxis Project Manager invokes, you can use the Pyxis Project Manager startup file. For example, to load the type registry *new_registry.rgy*, which is located under your user directory at */mgc/reg*, using your customized startup file, place the following commands in the startup file *your_home/mgc/startup/dmgr.startup*:

```
$set_active_window($open_types_window());
$load_registry("your_home/mgc/reg/registry.rgy");
$close_window();
```

Related Topics

[Problems with Type Registries](#)

Adding a Type with the MGC_TYPE_REGISTRY Shell Variable

Add type registries to the type manager through the MGC_TYPE_REGISTRY shell variable.

Procedure

1. Set the MGC_TYPE_REGISTRY variable to a series of type registry pathnames, each separated by a “:”, by entering the following at a UNIX System V shell prompt:

```
$ MGC_TYPE_REGISTRY=registry path 1: registry path 2
```

Remember to use full pathnames for each type registry. Also, you can specify any number of type registries. For example, to add registries `your_home/mgc/reg/my_defs.rgy` and `your_home/mgc/reg/my_2defs.rgy`, you use this command:

```
$MGC_TYPE_REGISTRY=your_home/mgc/reg/my_defs.rgy:your_home/mgc/reg/my_2defs.rgy
```

2. Verify that the value of MGC_TYPE_REGISTRY is correct, by entering the following at the shell prompt:

```
$ echo $MGC_TYPE_REGISTRY
```

The shell displays the current value of variable MGC_TYPE_REGISTRY. If this value is not correct, repeat step 1.

3. Export the MGC_TYPE_REGISTRY shell variable, by entering the following at the shell prompt:

```
$ export MGC_TYPE_REGISTRY
```

You must export the variable so that the Pyxis Project Manager, which runs in a process that is spawned from this shell, inherits and understands the value of MGC_TYPE_REGISTRY.

When you invoke the Pyxis Project Manager from the shell in which you set the value of MGC_TYPE_REGISTRY, the Pyxis Project Manager type manager inherits the value of MGC_TYPE_REGISTRY.

If you open subsequent shells from this shell, the newly opened shells also inherit the value of MGC_TYPE_REGISTRY, and you need not repeat the previous steps.

Related Topics

[Problems with Type Registries](#)

Appendix B

Pyxis Project Manager Standard Properties

The Pyxis Project Manager uses and assigns properties to design objects, references, and versions.

Design Object Properties..... 507

Design Object Properties

The Pyxis Project Manager uses properties to record specific information and to determine the proper behavior of certain operations.

Table B-1 lists the property names, the items to which the Pyxis Project Manager adds the property (the owner), and the property values. The value column describes the property's value and the operation during which the Pyxis Project Manager adds the property.

Table B-1. Design Object Properties Added by the Pyxis Project Manager

Property Name	Owner	Property Value
creating_tool	Design object reference	The name of the tool that created the reference. The Pyxis Project Manager can alter only those references that have this property and with a value of “project manager”. The only exception is that the Pyxis Project Manager’s “change references” command can change the pathname of any reference.
dme_config_ignore	Design object reference	None needed. The presence of this property prevents the “build configuration” command from traversing this reference when creating a configuration.
from_path	Design object reference	The original location of a design object that is part of a configuration. After a release or copy of a configuration, this property is added to the references between the configuration object and the members of the configuration.

Table B-1. Design Object Properties Added by the Pyxis Project Manager

Property Name	Owner	Property Value
from_version	Design object reference	The original version number of a design object that is part of a configuration (see “From_path”).
release_comment	Design object version	A comment about a particular version of a release.
release_date	Design object reference	A UNIX time integer value that designates the date when a design was last released.
release_date_string	Design object reference	A text string that designates the date when a design was last released.
released_to_path	Design object reference	The released location of a design object that is part of a configuration. After a design release, this property is added to the references between the configuration object and its entries.
released_by	Design object reference	The user login name of the shell process from which a configuration was most recently released.
tool_reference	Tool viewpoint reference	A string, either “qual” or “term”, that designates whether the reference points to the tool viewpoint’s qualification script or to its termination script.

Appendix C

Design Object Filesets

A design object's fileset is the combination of files and directories that compose that object.

The files hold either design data or metadata, which is data that describes the design object. The directory may contain other files or design objects. The Pyxis Project Manager treats the entire fileset as a single object. When you use the Pyxis Project Manager to move, copy, or delete a design object, that object's entire fileset is moved, copied, or deleted.

The Pyxis Project Manager relieves you from having to remember which files compose a design object. However, you occasionally want to list the contents of directories in the operating system shell, to view the filesets of design objects.

Data Files	509
Metadata Files	513

Data Files

Data files hold the design data for the aspect of the design that the design object represents.

In general, each data file's name has the following format:

name.suffix

where *name* is the name of the design object, and *suffix* is a string of characters that distinguishes one design file from another.

A design object's fileset may have several files, each with the same prefix, but each with different suffixes. As an example of a fileset, assume that you have a *schematic* design object called *fred* that is defined to consist of four files: *schem_id*, *.mgc_sheet.attr*, *.sgfx_1*, and *.ssht_1*. In the operating system shell, the fileset of design object *fred* would look like this:

```
schem_id
fred.ssht_1
fred.mgc_sheet.attr
fred.sgfx_1
```

Although these files might coexist in the same directory with many other files and directories, the Pyxis Project Manager recognizes these four files as being a *schematic* design object.



Tip: For instructions on determining a design object's fileset, refer to “[Getting Information about a Design Object](#)” in Chapter 3.

Required and Optional Files	510
Files Without Suffixes	510
Attribute Files	510
Version Files	511
Container Filesets	512

Required and Optional Files

Not all data files are necessarily required for the application or the Pyxis Project Manager to recognize the design object.

The files that are not required provide temporary or non-critical data for the design object. Assume that the *schematic* design object type defines the *.attr*, and *.schem_id* files as required and the *.ssht* and *.sgfx* files as optional. In that case, the following is a valid instance of type *schematic* named *fred*:

```
schem_id  
fred.mgc_sheet.attr
```

Related Topics

Files Without Suffixes	Version Files
Attribute Files	Container Filesets

Files Without Suffixes

Some design object types specify that a fileset member be recognized by name instead of by suffix.

For example, type *schematic* might specify that its instances consist only of a file called *fred*. In this case, when you navigate to a directory that contains a file called *fred*, the Pyxis Project Manager recognizes that file as a whole design object. Instead of displaying the generic file icon, the Pyxis Project Manager displays the icon for type *schematic*.

Related Topics

Required and Optional Files	Version Files
Attribute Files	Container Filesets

Attribute Files

The attribute file stores metadata for the design object. This metadata includes references, properties, and version information.

Attribute files are an important, permanent part of most design objects. The other, more temporary metadata files are discussed in “[Metadata Files](#)” in this appendix. Attribute files are not required by all design objects, but they are present in the fileset in the following cases:

- Versioned design objects always have an attribute file because the attribute file holds version information.
- Some container design objects must have an attribute file in order to be recognized as design objects.
- Unversioned design objects might not initially have an attribute file. However, as soon as you add metadata through your application, the application adds an attribute file to your design object's fileset.

The attribute filename has the following format where *type* is the type of your design object:

name.type.attr

When an attribute file is present for the *schematic* design object *fred*, the fileset looks like this:

```
schem_id  
fred.ssht_1  
fred.mgc_sheet.attr  
fred.sgfx_1
```

To quickly determine a design object's type, view the name of its attribute file in the operating system shell. The type name is always embedded between the design object's name and the “.attr” suffix.

Caution

 Although attribute files are in ASCII format, and as a result, can be edited from within the file system, **do not edit them**. When you edit an attribute file directly, you can corrupt the design object to which it belongs, so that it cannot be repaired.

Related Topics

[Files Without Suffixes](#)

[Version Files](#)

[Required and Optional Files](#)

[Container Filesets](#)

[Metadata Files](#)

Version Files

If a design object can have multiple versions, each version is part of the fileset. The version number is appended to the data file name, with the format *name.suffix_version*.

Assume that instead of being unversioned as in the previous examples, the *schematic* type specifies a design object that can have multiple versions. If our *schematic* design object *fred* contained versions 3 and 4, its fileset would look like this:

```
schem_id
fred.ssht_3
fred.ssht_4
fred.mgc_sheet.attr
fred.sgfx_
fred.sgfx_4
```

Note that the attribute file is not versioned; a single attribute file describes the metadata of all versions of the design object.

When you list the fileset in the shell, the number at the end of each filename indicates what versions that the design object has. The largest number is the current version number.

Related Topics

[Files Without Suffixes](#)

[Attribute Files](#)

[Required and Optional Files](#)

[Container Filesets](#)

Container Filesets

A container is a design object whose fileset includes a directory.

Although similar to a directory, a container may also include multiple files and directories in its fileset and has a unique, distinctive icon in the Pyxis Project Manager.

Mentor Graphics provides two kinds of containers:

- Containers that require an attribute file

These are usually directories that, as a result of having a unique type, have a unique icon in the Pyxis Project Manager. You can add certain types of these containers from within the Pyxis Project Manager.

- Containers that do not require an attribute file

These containers are combinations of directories and files that have unique suffixes. These files contain design data.

Assume that you have a container named *gate1* of type *schematic_container*. This container requires an attribute file that exists only to hold schematic design objects and to display a unique icon in the Pyxis Project Manager. Its fileset would look like this:

```
gate1
gate1.schematic_container.attr
```

Although this attribute file might not contain any references, properties, or version information, it is important, and you must not delete it. The Pyxis Project Manager uses this file to decide which icon to display for the container.

Now assume that you have a container named *cell2* of type *layout_container*. Further assume that this container does not require an attribute file, but it does require a data file with a *.design* suffix. Its fileset would look like this:

```
cell2
cell2.design
```

This design object can contain other design objects (inside the directory component of its fileset), would have a unique icon in the Pyxis Project Manager, and could carry design data in its fileset.

Related Topics

[Files Without Suffixes](#)

[Version Files](#)

[Required and Optional Files](#)

[Attribute Files](#)

[Adding a Container](#)

Metadata Files

Metadata is data that describes the design data. In other words, it is information about the current state of the design object and its relationship with other design objects.

The most important metadata file is the attribute file.

The following text describes the temporary files that appear in a design object's fileset and indicate the current state of the object. For example, these files indicate whether the design object is locked, is being edited, and so on.

As with any fileset member, you should not use operating system commands to directly manipulate these files. These files are explained in this section because you might see them when you are in the shell and because you might need to check for their presence when salvaging damaged design objects.

The following list describes the format and purpose of each of these files.

- **Frozen Version File**

Format: *name.type.fz_n*, where *n* is the number of the frozen version.

The frozen version file indicates that a particular version is frozen. For example, if version 3 of *schematic* design object *fred* is frozen, the fileset looks like this:

```
schem_id
```

```
fred.ssht_3
fred.ssht_4
fred.mgc_sheet.fz_3
fred.mgc_sheet.attr
fred.sgfx_
fred.sgfx_4
```

The *.fz* file appears when you use the Freeze Version command to freeze version number 3, and it remains until you use the Unfreeze Version command to unfreeze version number 3.

- **Lock File**

Format: *name.type.lck*

This file indicates that the design object is “locked,” which means that someone is currently editing it or that someone is reading it and wants to prevent others from editing it. Your application can apply two kinds of locks to a design object: read locks and write locks.

- **Read Locks.** When your application read-locks a design object, every user can read, but not edit, that design object. This lock is also known as a *shared* lock because several applications can read-lock a design object simultaneously. The object remains read-locked until all of the applications release their locks. Note that your application need not read-lock a design object in order to read it; the purpose of the read-lock is simply to prevent any user from editing the design object. Your application typically read-locks a design object when doing some operation during which the design object's data must not change.
- **Write Locks.** When your application write-locks a design object, you can edit that design object. In addition, others can read it, but no others can edit it or lock it. To edit a design object, the application must first write-lock it. This lock is also known as an *exclusive* lock because only one application can write-lock a design object, and when a design object is write-locked, no other application can lock it.

For an example of a lock file, assume that you are editing *schematic* design object *fred*. While it is locked, the design object's fileset looks like this:

```
schem_id
fred.ssht_3
fred.ssht_4
fred.mgc_sheet.fz_3
fred.mgc_sheet.attr
fred.sgfx_
fred.sgfx_4
fred.mgc_sheet.lck
```

In addition to applications locking design objects, you can lock design objects in two ways:

- Include the object in a configuration and lock the entire configuration.

- Use the `$$lock_object()` function, which allows you to lock a single design object.

You can determine if an object is locked and which user holds that lock by selecting a design object in the navigator and executing the popup menu item **Report > Object Info**.

If you see a `.lck` file in a design object's fileset, and if you determine that no application is using that design object, that object is damaged and you must use the **salvage_object** shell command to repair it.

- **Edit File**

Format: `name.suffix.ed`. This file appears for each fileset member on which your application is working.

This file holds edits to the design object until the application writes a new version. Each fileset member that the application can write can have a `.ed` file. If the application terminates abnormally before you write a new version, these files enable you to recover your edits, using the **salvage_object** command. This command saves a new version of the damaged design object, by using the data in the `.ed` files.

For an example, assume that you are editing *schematic* design object *fred* and have instructed the schematic editor to flush your edits out of your workstation's memory and onto the disk. The term “*flush*” appears often in this discussion; it means, “write your edits to the disk without creating a new version.” However, you have not yet instructed the schematic editor to write version 5 of *fred*.

The fileset looks like this:

```
schem_id
fred.ssht_3
fred.ssht_4
fred.ssht.ed
fred.mgc_sheet.attr
fred.mgc_sheet.fz_3
fred.mgc_sheet.lck
fred.sgfd_
fred.sgfd_4
fred.sgfd.ed
fred.mgc_sheet.attr.ed
```

Note that the design object is always write-locked when `.ed` files appear. If you see `.ed` files in a design object's fileset, and if you determine that no application is using that design object, that object is broken, and you need to use the **salvage_object** command to repair it.

- **New Edit File**

Format: `name.suffix.ed.new`. This file appears for each fileset member on which your application is working.

This file holds edits to the design object while your application is flushing your edits from memory to disk. When you flush your edits from memory, your application writes a *.ed.new* file for each member of the fileset on which it is working. When all of the *.ed.new* files are written, your application renames them to have *.ed* suffixes. With this method, if your application or network fails while writing your edits to the disk, your previously flushed edits are safe in the *.ed* files. If something does fail during a flush, you use the **salvage_object** command to write the safe, *.ed* edits to a new version of the design object.

Assume that you are editing *schematic* design object *fred* and have instructed the schematic editor to flush your edits to the disk. If you listed the directory in which *fred* resides while the schematic editor was flushing the edits, you might see this:

```
schem_id
fred.ssht_3
fred.ssht_4
fred.ssht.ed.new
fred.mgc_sheet.attr
fred.mgc_sheet.lck
fred.sgfx_
fred.sgfx_4
fred.sgfx.ed.new
fred.mgc_sheet.fz_3
```

Note that at this moment, the edits of only two of the fileset members are written to temporary files. You are more likely to see the edits of only two of these fileset members than all four *.ed.new* files (three data files and an attribute file) because as soon as all four are on the disk, they are renamed to *.ed*. The renaming operation happens so fast that you are not likely to see all four *.ed.new* files simultaneously.

If you see *.ed.new* files in a fileset for more than a few seconds, that design object is broken. The presence of *.ed.new* files indicates that no *.attr.ed* files exist. The **salvage_object** command is successful only when a *.attr.ed* file exists, so it never works on *.ed.new* data.

Related Topics

[Attribute Files](#)

[\\$\\$lock_object\(\)](#)

[Lock a Design Configuration](#)

[salvage_object](#)

Appendix D

Pyxis Project Manager Design Object Types

Pyxis Project Manager uses many design object types.

Design Objects

The column titled “V” in the table specifies whether the object is versioned (Y=Yes or N=No).

The icons shown are used in the Pyxis Project Manager iconic and list navigators, respectively, from left to right.

Table D-1. Pyxis Project Manager Design Objects

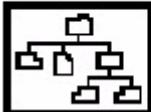
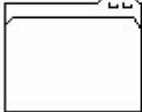
Design Object Type Name and Icons	Purpose of Design Object	Fileset	V
Dme_config_d0  	The Pyxis Project Manager's configuration object. This object stores the relationships between objects in a configuration, which is created as part of releasing a design.	my_obj.cfg my_obj.Dme_config_d0.attr	Y
Mgc_container  	A generic directory. The design object type supplies the generic folder icon in the Pyxis Project Manager's navigators and provides metadata for the directory object.	my_obj (directory) my_obj.Mgc_container.attr	N
Mgc_default_ascii_editor  	The tool viewpoint for the editor that appears in the tool window and that invokes when you double-click a file in the navigator.	Editor (directory) Editor. Mgc_default_ascii_editor.attr	N

Table D-1. Pyxis Project Manager Design Objects (cont.)

Design Object Type Name and Icons	Purpose of Design Object	Fileset	V
Mgc_file  	A generic file. The design object type supplies the generic file icon in the Pyxis Project Manager's navigators and provides metadata for the file object.	my_obj(file) my_obj.Mgc_file.attr	N

Appendix E

Design Object Types and Tool Icons

Design object types are identified by the Pyxis Project Manager based on fileset extensions and then displayed as unique icons that allow you to invoke the appropriate applications.

Design Object Types	519
Pyxis Custom Design Flow Tool Icons	520

Design Object Types

The column titled “V” in the table specifies whether the object is versioned (Y=Yes or N=No), where the version number is represented by a “#”.

Table E-1. Design Object Types

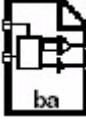
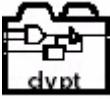
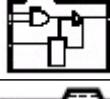
Design Object Type	Icon	Purpose of Design Object	Fileset (X is object name)	V
Eddm_ba_mgr		Backannotation data object.	default.Eddm_ba_mgr.attr default.ba_#	Y
Eddm_design_viewpoint		Design viewpoint. A container object that defines the design configuration rules and references backannotation objects.	default (directory) default.Eddm_design_viewpoint.attr default.dvpt_#	Y
Eddm_part		Object ties together all models associated with a component.	part.Eddm_part.attr part.part_#	Y
mgc_component		Component. A container object.	X (directory) X.mgc_component.attr	N
Mgc_container		A directory or directory with attribute file.	X (directory) X.mgc_container.attr	N
mgc_schematic		Schematic. Container for sheets. References and is referenced by mgc_sheet objects.	schematic (directory) schematic.mgc_schematic.attr schematic/schem_id	N

Table E-1. Design Object Types (cont.)

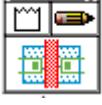
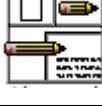
Design Object Type	Icon	Purpose of Design Object	Fileset (X is object name)	V
mgc_sheet		Sheet. Contains graphical and connectivity data for Pyxis Schematic sheets.	sheet1.mgc_sheet.attr sheet1.sgfx_# sheet1.ssht_#	Y
mgc_symbol		Symbol. Contains graphical symbol and properties.	X.mgc_symbol.attr X.smbl_#	Y
Simv_kernel_setup		Database created when saving QuickSim II kernel setup.	quicksim_setup (directory) quicksim_setup.Simv_kernel_setup.attr	N
Simv_log		ASCII file containing logfile information used in QuickSim.	X.log	N
Simv_misl		ASCII misl file.	X.misl	N
Simv_save_state		Database created when saving QuickSim II state information.	quicksim_state (directory) quicksim_state.Simv_save_state.attr	N
Simv_simview_setup		Common simulation (SimView) setup data.	simview_setup (directory) simview_setup.Simv_simview_setup.attr	N
Svdm_svdb		Simulation vector data model. Waveform database files used with simulation.	X.Svdm_svdb.attr X.dat_# X.wdb_#	Y
Tf_tfile_do		Compiled technology file.	X.Tf_tfile_do.attr X.tecf_#	Y

Pyxis Custom Design Flow Tool Icons

The Pyxis Project Manager examines a design structure and compares design objects to registered tool viewpoints. A tool viewpoint contains a tool icon for that tool.

Table E-2 lists the tool icons used with Pyxis Custom Design flow designs.

Table E-2. Pyxis Custom Design Flow Application Tool Icons

Application Name	Tool Icon	Tool Icon Name
Pyxis Schematic		da_ic
Pyxis Design Viewpoint		DVE_IC
Pyxis Layout		ic
Notepad		Editor_ic

Appendix F

Releasing Latched Data in Pyxis Project Manager

The Pyxis Project Manager supports the latching of data by providing you with the ability to build and release latched viewpoints.

Latching is described in detail in the section “Design Latching” in the *Design Viewpoint Editor User’s and Reference Manual* on SupportNet. For specific information on the commands that you use to latch design viewpoints, refer to the section “Function Dictionary” in the *Design Viewpoint Editor User’s and Reference Manual* on SupportNet.

Before reading this section, you should be familiar with the concepts of building and releasing a configuration that is not latched. These concepts are discussed in the sections “[Releasing a Design Configuration](#)” on page 266 and “[Creating a Configuration](#)” on page 118.

Constraints for Releasing Latched Viewpoints	523
Building and Releasing a Configuration in Latched Mode	524

Constraints for Releasing Latched Viewpoints

Latching and saving a design viewpoint in the Design Viewpoint Editor (DVE) creates a current reproducible *snapshot* of your design viewpoint by latching the specific version of each design object that the design viewpoint currently references.

This allows you to have a “frozen” snapshot of your design viewpoint to work with, while others on your design team continue to evolve the design. Each time that you invoke an application on the latched design viewpoint, the latched versions of the referenced objects are used to reproduce the design viewpoint exactly as it was when you first latched it. Because the design object versions that are referenced by the latched design viewpoint cannot be deleted, you can be sure that no part of your design viewpoint is accidentally deleted.

Note



You cannot explicitly delete design data in the Pyxis Project Manager which has been latched by DVE. That is, you cannot delete latched data by using the `$delete_object()`, `$delete_version()`, or `$prune_design_hierarchy()` functions. In the Pyxis Project Manager, you can only copy or release latched design data. However, you can delete latched design data that is contained within an unlatched container. For example, if latched data is contained within an unlatched directory and you execute the `$delete_object()` function on the directory, the latched data is also deleted.

When you are ready to release your latched design viewpoint, you can release your data from either the navigator or the configuration window. You use the same Pyxis Project Manager build and release commands that you use for releasing unlatched data. However, releasing *latched* data in the Pyxis Project Manager imposes several additional constraints.

If you release latched data in the navigator window, you must meet the following constraints:

- The design viewpoint must have been created and latched in the Design Viewpoint Editor (DVE).
- The design viewpoint must be the only selected object in the navigator window when you perform the release operation.

If you release latched data in the configuration window, you must meet the following constraints:

- The design viewpoint must have been created and latched in the Design Viewpoint Editor (DVE).
- The design viewpoint must be the only primary entry in the configuration prior to and during the build operation.
- In the configuration window, you can add entries to the configuration after you have performed a build on the latched design viewpoint, as long as you do not perform another build operation. If you rebuild the configuration after adding additional entries, the configuration is no longer latched.

If any of the above conditions are violated in either the configuration or the navigator window, the Pyxis Project Manager treats the design viewpoint as an unlatched object and builds and releases your data according to the normal configuration methodologies.

After you release a latched design viewpoint, the released design viewpoint in the destination directory does not maintain the latched state of the source design viewpoint. To latch the released design viewpoint, you must return to DVE and perform a latch operation on the released design viewpoint.

Related Topics

[Creating a Configuration](#)

Building and Releasing a Configuration in Latched Mode

Normally, the build operation traverses the containment and reference hierarchy of the design viewpoint and adds a version of each referenced design object to the configuration as a secondary entry.

When a reference's state is *current* (referred to as a current reference) the current or latest version of the referenced design object is added to the configuration. When a reference's state is *fixed*, a previous version of the referenced design object is added.

When a build is performed in *latched mode*, a current reference is interpreted to mean the specific version of the design object that was referenced at the time the design viewpoint was latched. Therefore, in latched mode, each version of each design object that was referenced at the time the design viewpoint was latched is added to the configuration as a secondary entry.

Latched Mode in the Configuration Window	525
Latched Mode in the Navigator Window	525

Latched Mode in the Configuration Window

When you add the latched design viewpoint to the configuration as the only primary entry, you can build the configuration and then release it to a destination directory.

Building and releasing a latched design viewpoint in the Pyxis Project Manager configuration window according to the constraints described previously, causes the Pyxis Project Manager to operate in *latched mode*. To latch your design, utilize the **Edit>Latch Version>Latch Version...** menu item. An hourglass appears in the configuration window during the *processing* of the design's latching. When the latching process is completed, the hourglass disappears.

Although, only one primary entry can exist in the configuration window prior to and during a build operation, entries can be added to the configuration window after the latched design viewpoint has been built, as long as another build operation is not performed prior to the release operation. If an entry with a latched version corresponding to the viewpoint is added to the configuration after the build, the latched version of the object is used. If an entry is added to a configuration after the build is performed and it does not have a latched version that corresponds to the viewpoint, the current version of the object is used. If the configuration is rebuilt after another entry has been added, the configuration is no longer latched. For additional information on latching in the configuration window, refer to the *Design Viewpoint Editor User's and Reference Manual* on SupportNet.

Related Topics

[Latched Mode in the Navigator Window](#)

Latched Mode in the Navigator Window

In the navigator window, you can only select a single design viewpoint to release.

When you release a latched design viewpoint from the navigator window, a build is automatically performed by the release operation. Releasing a latched design viewpoint in the navigator window according to the constraints described previously causes the release to operate in latched mode. If you select more than one object to release in the navigator, the

release operation does not perform in latched mode, even when one or both of the objects are, themselves, latched.

Related Topics

[Latched Mode in the Configuration Window](#)

Appendix G

User Interface and Scopes

Pyxis Project Manager uses windows and userware scopes to help you perform particular tasks.

You should have knowledge of userware scopes and intend to customize the Pyxis Project Manager. For detailed information on userware scopes, refer to the *AMPLE for Pyxis User's Manual* on SupportNet.

Windows and Scopes [527](#)

Windows and Scopes

Each of the Pyxis Project Manager window types has a specific set of userware scopes.

For instance, in a configuration window, you perform configuration management operations, and in a reference window you work with references. A *userware scope* is a portion of the environment in which a function has a particular meaning. The userware scopes associated with a particular window type are available when the window is active.

The scopes in each of the Pyxis Project Manager window types are arranged in a hierarchy that is created by the application developer. The end-user cannot change the order of the scopes in this hierarchy or add new scopes to the hierarchy. However, the end-user can add new functions to scopes in the hierarchy or modify existing functions in the scope hierarchy.

When a function is called, the scope hierarchy of the active window is searched until the function is found. If the same function name exists in more than one scope, the first function found matching the specified name is executed.

Each scope has a corresponding AMPLE file that contains all the AMPLE userware for that scope. These AMPLE files are stored in the Mentor Graphics Software Tree. The AMPLE files for scopes common to all Pyxis Custom Design flow applications, including the Pyxis Project Manager, can be found in the following directory:

`${MGC_HOME}/shared/pkgs/ele_ic/userware/${LANG}`

The AMPLE files for scopes specific to the Pyxis Project Manager can be found in the following directory:

`${MGC_HOME}/shared/pkgs/dmgr_ic/userware/${LANG}`

In the following example, the pathname to the scope file for the ovl_area scope is based on the following conditions:

- The MGC_HOME shell environment variable is set to “/usr2/mgc_tree”.
- The LANG shell environment variable is set to “En_US”, (which resolves to En_na in the *mgc_lang_map* file), as shown below.

/usr2/mgc_tree/shared/pkgs/dmgr_ic/userware/En_na/ovl_area.ample

Note

 NOTE: For detailed information on the MGC_HOME and LANG shell environment variables, refer to the *AMPLE for Pyxis User's Manual*.

Some scopes are placed in the scope hierarchy of more than one window type and therefore the functions in that scope can be executed from more than one window type. For example, the \$close_session() function, in the dmgr_session_window scope, can be executed from all Pyxis Project Manager windows. Other scopes are available in only one particular window type and therefore the functions in that scope can be executed from only one window type. For example, the \$delete_reference_property() function, in the dmgr_reference_model scope, can only be executed from a reference window.

Related Topics

[AMPLE for Pyxis User's Manual](#)

[\\$ask_scope_frame_name\(\)](#)

Appendix H

Managing Location Maps

System Managers and other job roles may be responsible for general network and software setup such as location maps.

 **Note** The information on location maps in this appendix does not apply to location maps used with Pyxis Project Manager's Project Navigator.

Design Management with Location Maps	529
Location Map Description	532
How the DDMS Uses the Location Map	538
Location Map Creation	542

Design Management with Location Maps

As a system manager, you have a number of tasks to perform in relation to location maps:

- Managing location maps
- Controlling environment variables
- Administering networks

Master Location Maps	529
Project Location Maps	530
Individual Location Maps	531

Master Location Maps

As a system manager who manages network resources, you need to maintain a master location map that identifies the soft prefixes and environment variables being used by all projects.

A master location map has two purposes:

- Avoids conflicts between soft prefixes used by different projects.
- Provides a central location map that servers (such as printers and plotters) can use to properly identify design data files, independent of specific projects.

The master map is a master list of approved soft prefixes and their locations on the network. A master map might include other project maps through the use of multiple INCLUDE lines.

Master location maps potentially differ from project location maps in three ways:

1. Project location maps contain a subset of master location map entries.
2. Hard pathnames could differ between the master location map and project location maps (though they should point to the same objects).
3. Some projects might use different copies of the same libraries (different hard pathnames) to prevent network bottlenecks on large projects.

To successfully administer a master location map, you must establish and maintain open communication with engineering project leaders and application users. The following suggestions may assist you in maintaining a master location map:

- Work with project managers to assign unique soft prefixes to each project.
- Avoid duplicate user-created soft prefix names with different hard pathname definitions.
- Ensure that all project managers and team leaders know the location of the master location map, so they can read it and copy it for their own use. The master map should have read and execute permissions only. Only the system manager or person assigned to the task should have rights to change the master map.
- Keep a list of project location map administrators or project leaders. You need to tell them about location map issues and changes.
- Keep project managers informed about network changes and any libraries or other components that you move.
- Use the master location map, or a subset of the master location map, for shared servers. Each shared server process (such as a printer or plotter) must be able to read the project data and associated libraries. The server reads a location map into memory each time it starts a task, which allows you to change the location map to reflect data access changes without restarting your servers. The server picks up all changes in the map each time it starts a job. By making sure a server uses a master location map, you permit the server to use any soft prefix available on your network when it tries to find data.
- Consider creating duplicate copies of libraries and other components for redundancy and alleviating network traffic.

Related Topics

[Project Location Maps](#)

[Individual Location Maps](#)

Project Location Maps

Specific design projects might require their own location maps.

As a system manager, you need to work with project managers and team members to properly configure and maintain project location maps. The steps below provide a logical sequence for the creation of project location maps:

1. The project manager works with the system manager to derive a suitable project soft prefix, such as \$COMPANY_PROJ_X. An identifying prefix, such as \$COMPANY_PROJ_X, helps prevent conflicts with soft prefixes used on other projects.
2. The project manager identifies a list of soft prefixes for the design directories in their project, as well as the libraries they need to use.
3. The project manager submits these names to the system manager, who checks the soft prefixes against the master location map, and updates the master map with the new soft prefixes.
4. If there are soft prefix conflicts, the system manager discusses and resolves the conflicts with the project manager.
5. If project soft prefixes point to the same hard pathnames as existing soft prefixes, the system manager resolves the duplications with the project manager.
6. The project manager works with the system manager to ensure the project location map references the appropriate libraries. If duplicate libraries exist, the system manager assists the project manager to balance the network load.
7. The project manager creates a project location map that is a subset of the master location map.
8. If a single project map, with hard pathnames, is common to all project members, the project manager stores the shared map in a specific location.

Related Topics

[Master Location Maps](#)

[Individual Location Maps](#)

Individual Location Maps

If individual users create private location maps with personal soft prefixes that are not registered in project or master location maps, such users may encounter problems:

- They may prevent servers, such as printers and plotters, from operating properly. Servers are not able to locate the design data.
- They cannot share their data with other users, because other users cannot access the soft prefixes and hard pathnames contained in the individual location map.
- The designs they create run the risk of being unusable.

In general, a system manager should discourage individual users from creating their own location maps with personal soft prefixes, unless such users work alone on self-contained designs.

Related Topics

[Master Location Maps](#)
[Location Map Description](#)

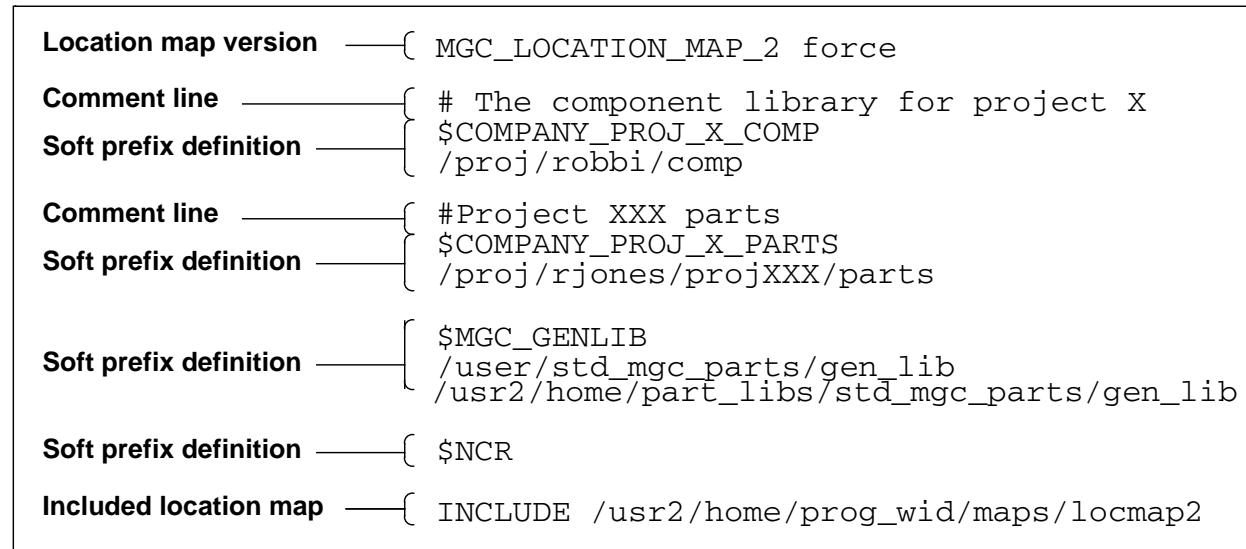
[Project Location Maps](#)

Location Map Description

The lines of a location map file describe the location map version, define the network location of objects denoted by soft prefixes, and list pathnames to other included location maps.

A location map can also include comment lines, which are ignored by Mentor Graphics software.

Figure H-1. Parts of a Location Map



Location Map Version	532
Comment Lines	533
Soft Prefix Definitions	534
Included Location Maps	537

Location Map Version

The location map file's first line is always MGC_LOCATION_MAP_x, where x is equivalent to the location map file's version as follows:

- MGC_LOCATION_MAP_3: This was introduced in the 2008.2 software version, and is fully supported in later software versions.
- MGC_LOCATION_MAP_2 and MGC_LOCATION_MAP_1: These are supported in software versions prior to the 2008.2 software version.

Note



Software Version 2008.2 or later software can recognize all location map versions.

Version 2008.2 or later can read MGC_LOCATION_MAP_3. When using this location map version, the last hard path is used for duplicate soft prefixes.

The MGC_LOCATION_MAP_x version header can contain one or more spaces and the string “force” as in the following example:

```
MGC_LOCATION_MAP_2 force
```

The “force” string converts hard pathnames to soft pathnames and, if unsuccessful, returns an error. If you omit the “force” string, then a hard pathname is used if Mentor Graphics software cannot convert a hard pathname to a soft pathname. Use the “force” string to prevent hard pathnames from being stored in design object references.

Related Topics

[Comment Lines](#)

[Soft Prefix Definitions](#)

[Included Location Maps](#)

Comment Lines

The pound sign (#) is the comment character for a location map, just as it is for a shell script.

The following is an example of a comment line in a location map:

```
# The part library for Project X
```

In a comment line, everything on the line following the pound sign is ignored.

Related Topics

[Location Map Version](#)

[Soft Prefix Definitions](#)

[Included Location Maps](#)

Soft Prefix Definitions

Each location map contains one or more soft prefix definitions. Your location map likely contains several of these.

A soft prefix definition consists of one line that defines the soft prefix, followed by zero, one, or several subsequent lines denoting the hard pathnames to the location of the object denoted by the prefix. If multiple hard pathnames follow a soft prefix, they must all be paths to the same object.

The following example shows three soft prefix definitions. Note how each definition differs in the number of lines that follow the soft prefix. Each of these variations are explained in the following pages.

```
$COMPANY_PROJ_X_PARTS
/proj/rjones/projXXX/part
$MGC_GENLIB
/user/std_mgc_parts/gen_lib
/usr2/home/part_libs/std_mgc_parts/gen_lib
$NCR
```

Any soft prefix definition must adhere to certain rules, as described in the following subsections.

Length Limit

A location map has an upper length limit of 1024 characters for both soft prefixes and hard pathnames. But, keep in mind that any soft prefix or hard name length must be compatible with the length limits for your operating system. If several hosts with different operating systems are using the same location map, pathname lengths should not exceed the maximum for the most restrictive operating system length.

In addition, any lines defining soft prefixes or hard pathnames cannot contain a blank space at the beginning of the line. If a blank space exists, an error message occurs.

Rules for Soft Prefixes

The two main soft prefix rules involve:

- What you name a soft prefix
- How you identify a soft prefix as a library

Naming Rules

Each soft prefix must begin with a dollar sign (\$) to distinguish it from an ordinary pathname. The dollar sign (\$) that begins the soft prefix must be the first character on a line.

Reserved soft prefixes for Mentor Graphics parts libraries begin with “\$MGC_” The following shows the soft prefix for the Mentor Graphics generic parts library, gen_lib:

```
$MGC_GENLIB
```

Soft prefixes must meet the same standards as shell environment variables, which means that they must not contain shell special characters, such as a period (.) or question mark (?), or an error can occur. Mentor Graphics software also checks the following conditions and returns an error message if they are not met:

- The soft prefix cannot be a null string.
- The first character in the soft prefix name that follows the dollar sign should be an alpha character or an underscore.
- Additional characters must be alphanumeric characters or underscores.

Type Identifiers

An optional type identifier can be used to define the soft prefix as a certain type. Currently, the only type identifier allowed is LIBRARY, as follows:

```
$<soft_prefix> -t LIBRARY
```

The **-t LIBRARY** type identifier specifies that the given softname denotes a library. The Pyxis Project Manager can then use the type identifier to identify libraries for configuration and release operations. Some Pyxis Project Manager commands allow this type of library filtering instead of, or in addition to, filtering by pathname.

Note

 The **-t LIBRARY** switch requires that a hardpath name is supplied. Soft pathnames are not supported with this switch. If a soft pathname is supplied then any environment variables that would overwrite the soft prefix are not available.

If a location map contains soft prefixes that use a type identifier, the version of the map must be a “2” or a “3”.

Rules for Hard Pathnames

Hard pathnames must begin with the customary slash (/) that indicates a pathname. A hard pathname is mapped to the closest preceding soft prefix. The slash (/) that begins a hard pathname must be the first character on a line. The following example shows the hard name that corresponds to \$COMPANY_PROJ_X_COMP:

```
$COMPANY_PROJ_X_COMP
/proj/robbi/comp
```

Any pathnames in a location map should be equivalent to the pathname you would use in a **cd** command to get to that location. Mentor Graphics strongly advises against placing pathnames with double slashes in a location map. A location map with a double slash pathname is not interoperable with UNIX systems. Rather, you might want to show the single slash equivalent and comment the double slash pathname, as follows:

```
$MGC_GENLIB
/usr/gen_lib
# //dodger/local_user/gen_lib
```

Up to 19 multiple hard pathnames can follow a single soft prefix. If multiple hard pathnames exist for a single soft prefix, they must all be alternate paths to the same object. In an NFS based-environment, the hard path that always resolves from anywhere within the network should appear first, followed by other pathnames to get to the same object.

Only the first hard pathname is used to resolve existing design references when the location map is read into memory; however, any of the qualified hard pathnames can be used to produce a soft prefix when a user enters new design references. The following example shows a soft prefix definition with two pathnames to the same directory:

```
# Project XXX Parts
$COMPANY_PROJ_X_PARTS
/home/rjones/projXXX/parts
/project/projXXX/parts
```

Because the pathname */home/rjones/projXXX/parts* can be resolved from any system in the network, including the local system, it appears first in the list of hard pathnames.

Soft Prefixes Without Hard Pathnames

A soft prefix does not have to have a hard pathname following it. These type of entries are placed in the location map to tell the design data management system (DDMS; the Pyxis Custom Design flow component that manages design data) that an environment variable of the given name exists.

The following example shows a soft prefix with no hard pathnames:

```
$NCR
```

With \$NCR in the location map, you can use the value of the \$NCR environment variable to update your location map every time the NCR library is moved. The system manager sets the \$NCR environment variable at the system level, or the user sets it at the shell level, which allows the \$NCR location map entry to return the correct pathname value when accessed.

If a soft prefix has no associated hard pathname and no corresponding environment variable exists, Mentor Graphics software returns an error message to the calling application.

Related Topics

[Location Map Version](#)
[Included Location Maps](#)

[Comment Lines](#)
[How the DDMS Uses the Location Map](#)

Included Location Maps

The optional INCLUDE keyword in a location map allows the inclusion of the contents of another location map by reference.

When an application reads a location map into memory that contains an INCLUDE statement, the in-memory location map is produced just as if the included location map had been typed into the original location map at the position of the INCLUDE statement.

By convention, the INCLUDE line is the last line in the location map file. However, an INCLUDE line can actually be placed anywhere in your location map file. Because Mentor Graphics software searches a location map from top to bottom, the placement of an INCLUDE entry could be important in your network, especially if the local location map and included location map contain different definitions for the same soft prefix.

If there are duplicate soft prefixes as a result of an included location map(s), they result in the following:

- the first soft prefix definition is used
- the other definitions are ignored
- an error results

Because of the above, the placement of any INCLUDE lines should be deliberate and well thought out.

The format of an INCLUDE entry in the location map is as follows:

```
INCLUDE <hard pathname to the location map to include>
```

If your location map includes several other maps, there might be multiple INCLUDE lines in the file. If a location map contains a line using the INCLUDE keyword, the version of the map must be a “2” or a “3”. An INCLUDE line can reference version 1 or version 2 location maps.

Related Topics

[Location Map Version](#)
[Soft Prefix Definitions](#)

[Comment Lines](#)

How the DDMS Uses the Location Map

The Pyxis Custom Design flow component that manages design data is called the Design Data Management System, or DDMS.

Location Map Search Hierarchy	538
The Location Map and Existing References	540
The Location Map and New References	541

Location Map Search Hierarchy

When a user invokes an application, the application relies on the DDMS to follow the search hierarchy to find a location map.

When the DDMS finds the map, it stops searching and loads the information contained in the location map into memory, where it is used for resolving references during the design process. Referring to an in-memory location map during the design process is much faster than attempting to resolve pathnames using a location map stored at another network location.

The following is the search hierarchy that the DDMS uses to find a location map (a flowchart of the search hierarchy appears in Figure H-2):

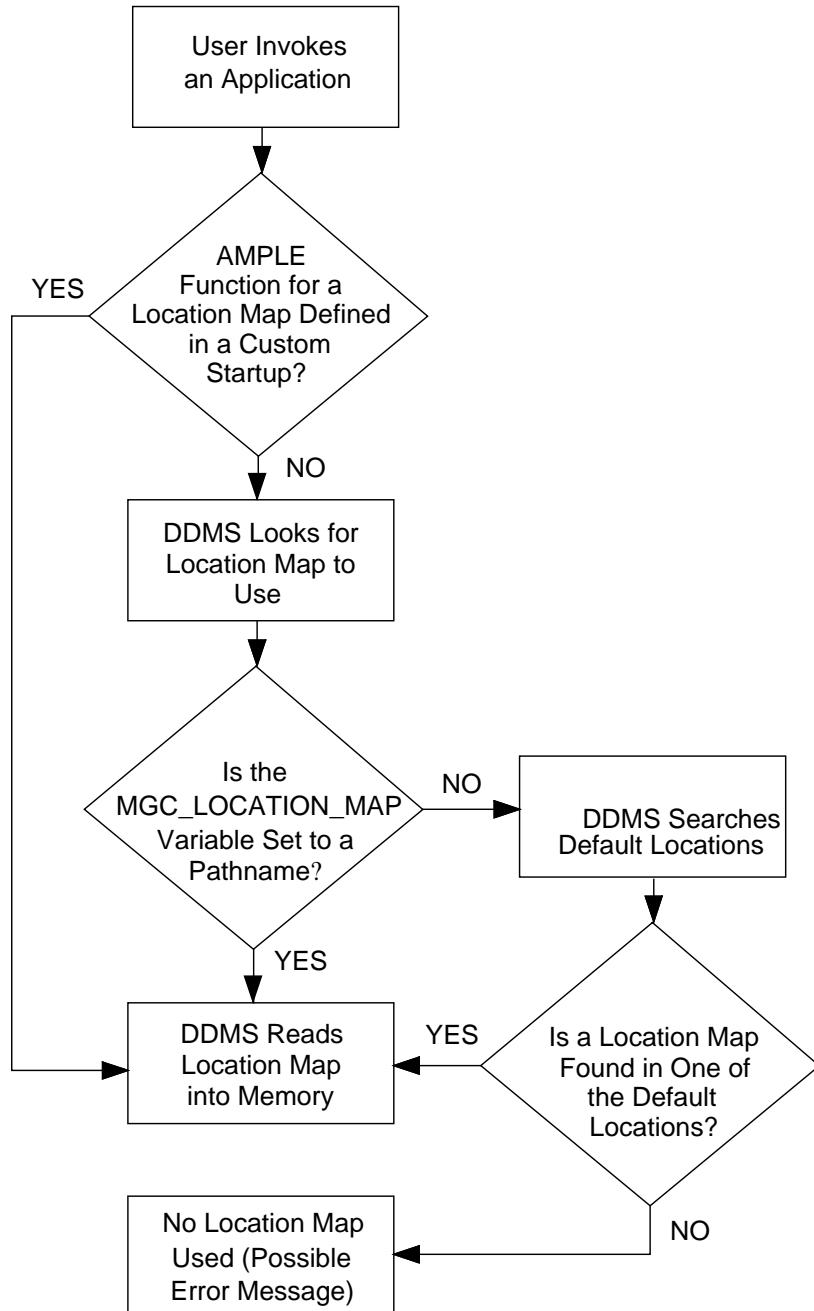
1. An AMPLE function that points to a valid location map exists in a custom application startup file. If such a function exists, the DDMS loads the location map into memory based on the value returned by the AMPLE function.
2. An MGC_LOCATION_MAP environment variable, whose value is set to a pathname. If set to a pathname, the value of the MGC_LOCATION_MAP environment variable tells the DDMS where to find the appropriate location map. If the variable is set to the string “NO_MAP”, then DDMS assumes no location map should be used and stops searching. MGC_LOCATION_MAP can be set in a user’s login files or application startup shell scripts.
3. A series of default locations. If the MGC_LOCATION_MAP variable is not set, the DDMS searches the following default locations for a location map:

```
./mgc_location_map
$HOME/mgc/mgc_location_map
$MGC_HOME/etc/mgc_location_map
$MGC_HOME/shared/etc/mgc_location_map
```

If there is no AMPLE function specifying a location map in the custom startup file, the MGC_LOCATION_MAP is not set, or there is no location map in any of the default locations, the DDMS assumes that you do not want to use a location map. This situation could result in an error if a design uses soft prefixes or references Mentor Graphics libraries that are not otherwise defined through an environment variable.

A simple way to determine the location map that is found and used by an application is through the ddms_which_map command. The command goes through the location map search rules and returns the full pathname to the location map. If no location map is used, ddms_which_map displays a message to that effect. If the returned pathname provided is not an existing pathname, “Not found” is printed after the file name.

Figure H-2. Reading a Location Map Into Memory



Related Topics

[The Location Map and Existing References](#)

[The Location Map and New References](#)

The Location Map and Existing References

When the application requires interpretation of an existing soft prefix in a design reference, the DDMS uses the location map stored in local memory to resolve references as follows:

1. The DDMS checks the contents of the in-memory location map for the first soft prefix that corresponds to the soft prefix in the design reference.
2. After finding a soft prefix, the DDMS compares that soft prefix against any environment variables that might exist with the same name.
 - o If the DDMS finds an environment variable with the same name as a soft prefix in the location map and the value of the environment variable differs from the first hard pathname associated with the soft prefix, it ignores the hard pathnames associated with the soft prefix in the location map file. Instead, it uses the value of the environment variable for the hard pathname.

For example, if the location map entry is:

```
$COMPANY_PROJ_X_COMP  
/mnt/jack/comp
```

and if the value of the environment variable \$COMPANY_PROJ_X_COMP is:

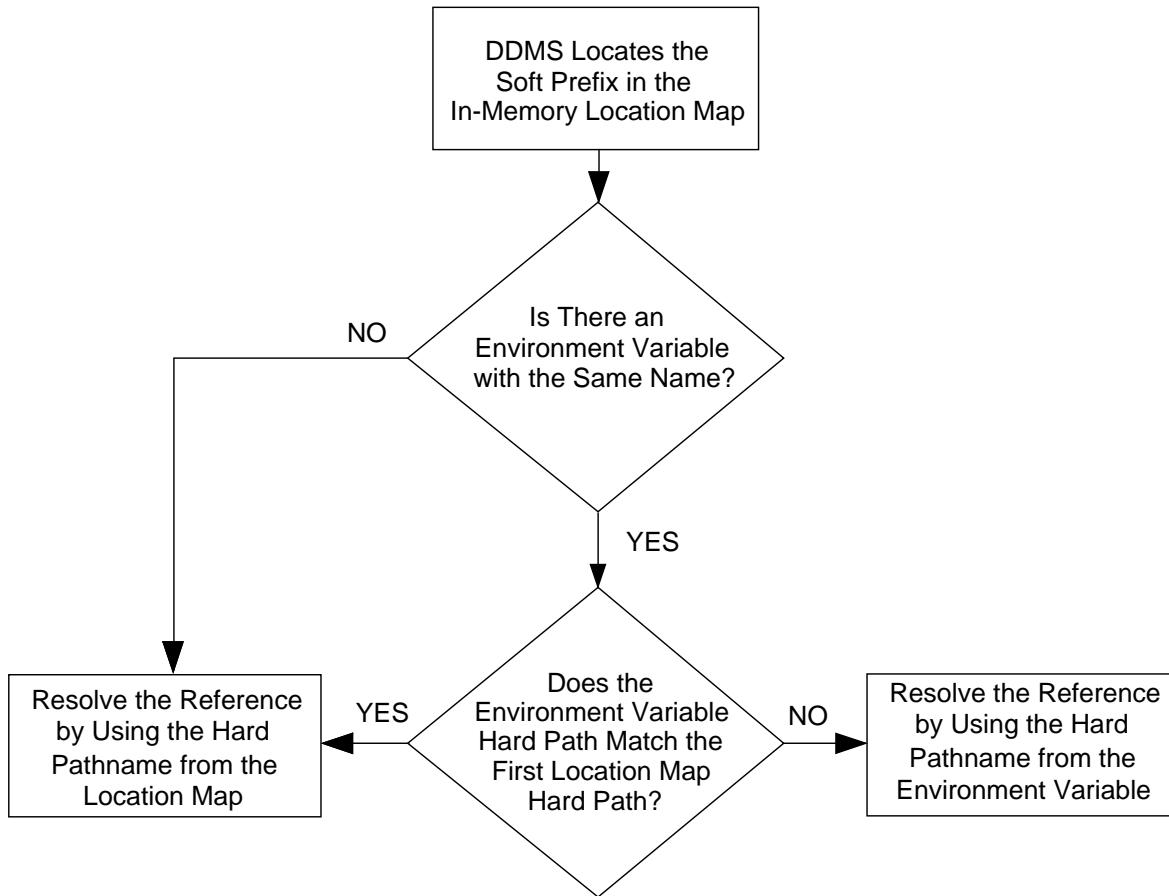
```
/proj/robbi/comp
```

the DDMS attaches the value of */proj/robbi/comp* to the in-memory map and discards the other hard pathnames in the location map corresponding to \$COMPANY_PROJ_X_COMP.

- o If there is no environment variable with the same name as the soft prefix in the location map, the DDMS resolves the reference using the value of the first hard pathname following the soft prefix in the in-memory location map.

Since the value of an environment variable overrides the location map entry, you need to carefully control the use of environment variables.

Figure H-3. Location Map Resolves an Existing Design Reference



Related Topics

[Location Map Search Hierarchy](#)

[The Location Map and New References](#)

The Location Map and New References

When you create a new design reference from within a Mentor Graphics application, the DDMS uses the in-memory location map to handle it in one of several ways:

- **If you enter a hard pathname that begins with a slash (/),** the DDMS tries to convert it to a soft pathname before storing it as part of the design. The DDMS compares the pathname you enter against the hard pathnames in the location map. If it finds a match, the first soft prefix in the location map that corresponds to the hard pathname is substituted for the portion of the pathname that matches.

For example, if you enter `/proj/robbi/comp/a_comp` as a pathname to a component, the DDMS searches the map for the first matching name that is a qualified path, comparing

the hard pathname you entered against complete path elements. Both */proj/robbi/comp* or */proj/robbi* are considered matches, but */proj/robbi/com* is not. The slash (/) character is the delimiter for path qualification and all elements of the path following the slash must match exactly. If the DDMS finds a match, such as */proj/robbi/comp*, then it substitutes the attached soft prefix, such as *\$COMPANY_PROJ_X_COMP*, for that portion of the hard pathname. The DDMS ignores subsequent matching pathnames in the location map file. The reference then becomes stored in the design database as *\$COMPANY_PROJ_X_COMP/a_comp*.

- **If you enter a pathname that begins with a dollar sign (\$),** the DDMS compares the first element in the pathname with the soft prefixes in the location map. If no matching soft prefix is found, the DDMS returns an error. Therefore, you cannot create a reference using an environment variable without a matching soft prefix listed in the location map.
- **If you enter a pathname that does not begin with either a slash (/) or dollar sign (\$),** the DDMS assumes that it is a relative pathname. The DDMS converts relative pathnames to absolute pathnames by pre-pending the Mentor Graphics current working directory (MGC_WD) to the relative name and flattening any ./ or ./ sequences. The Mentor Graphics current working directory is not equivalent to the shell's current working directory. The absolute path is then processed as described in Step 1, if the value of the current working directory is a soft prefix.

Related Topics

[Location Map Search Hierarchy](#)

[The Location Map and Existing References](#)

Location Map Creation

Creating a location map typically involves copying the template location map created during installation and adding additional lines denoting special soft prefixes or other location maps to include.

Template File Location	542
Guidelines for Creating Soft Prefixes	543
Determining Existing Hard Pathnames and Soft Prefixes	544
Controlling Environment Variables	545
Location Map Network Administration Examples	547

Template File Location

When you install part libraries, the Install program builds a template location map and places it at *\$MGC_HOME/shared/etc/app/base/mgc_map_template*

Entries in that template map are based on the names and locations of the libraries you installed with the Install program.

You can then copy and customize this map as needed to include special soft prefixes for your site or the pathnames to other location maps you want to include. You can also use the “Configure Services” window of the Install program to copy the template location map. Do not leave a location map in the template location, as it will be overwritten the next time you install software.

Related Topics

[Guidelines for Creating Soft Prefixes](#)

[Determining Existing Hard Pathnames and Soft Prefixes](#)

[Controlling Environment Variables](#)

[Location Map Network Administration Examples](#)

Guidelines for Creating Soft Prefixes

You may want to make your soft prefixes conform to the suggested guidelines presented in this section. While not following these guidelines should not break design references, adhering to the suggested guidelines can make your map easier to read and improve look up performance.

Prefix Names

Mentor Graphics suggests the following guidelines when creating your own soft prefix names:

- Use a company name or acronym to differentiate your company's soft prefix from Mentor Graphics soft prefixes. The characters “MGC” are reserved for Mentor Graphics part libraries.
- Use a project name or acronym to differentiate soft prefixes from those of other projects.
- Use a descriptive name or acronym to make the prefix readily intelligible.

How Much a Soft Prefix Encompasses

To limit map size and execution time, each soft prefix in a location map should represent a significant portion of a project directory, preferably one containing subdirectories that are likely to be moved or copied together. For example, if you need to reference a schematic in `/proj/rjones/projXXX/parts`, it is more useful to assign the `$COMPANY_PROJ_X_PARTS` soft prefix to the entire `parts` directory than it is to create a separate soft prefix for the single schematic. If you need to access `/proj/rjones/projXXX/parts/schematic_a`, your reference then becomes `$COMPANY_PROJ_X_PARTS/schematic_a`.

Placement in the Location Map

Since the contents of a location map are searched from top to bottom, you should put definitions for more frequently used soft prefixes near the top of the map file to enhance performance. The DDMS accepts the first soft prefix that corresponds to the specified hard pathname, although all soft prefixes are read into memory when you invoke an application.

Multiple Prefixes to the Same Location

Do not use multiple soft prefixes for the same hard pathname. Though such multiple prefixes do not impair location map function, it is an inefficient use of the location map, and it is high in maintenance overhead.

Overlap of Soft Prefix Definitions

If possible, avoid using soft prefixes that reference subdirectories of another soft prefix. If you must use a soft prefix that references a subdirectory of another soft prefix, list the subdirectory definition first. If you do not, the subdirectory is never used, unless you explicitly specify it when creating a reference. For example, if you use the \$COMPANY_PROJ_PARTS soft prefix to refer to project X's parts libraries and the \$PROJ soft prefix to refer to the main project directory, list \$COMPANY_PROJ_PARTS first in the location map, as follows:

```
$COMPANY_PROJ_PARTS
/proj/rjones/projX/parts
$COMPANY_PROJ
/proj/rjones/projX
```

Related Topics

[Template File Location](#)

[Controlling Environment Variables](#)

[Determining Existing Hard Pathnames and Soft Prefixes](#)

[Location Map Network Administration Examples](#)

Determining Existing Hard Pathnames and Soft Prefixes

If you have a hard pathname to a design or component and want to know if a soft pathname exists, use the \$\$get_soft_name() function.

This function takes a hard pathname or a list of hard pathnames and returns the related soft pathnames. For example, to return the soft prefix corresponding to the pathname */projectx/design1/nand*, you would type:

```
$$get_soft_name( "/projectx/design1/nand" )
$MYCO_PROJX DESIGN1
```

If you have soft pathnames and want to know the associated hard pathnames, use the \$\$get_hard_name() function. This function takes a soft pathname or a list of soft pathnames and returns the related hard pathnames. For example, to find out the hard pathname associated with the soft prefix \$MYCO_PROJ_DESIGN1, you would type:

```
$$get_hard_name( "$MYCO_PROJX DESIGN1" )
/projectx/design1/nand
```

Related Topics

Template File Location	Controlling Environment Variables
Guidelines for Creating Soft Prefixes	Location Map Network Administration Examples

Controlling Environment Variables

The two variables related to location maps that must be set in a user's environment are MGC_LOCATION_MAP and MGC_WD.

MGC_LOCATION_MAP	545
MGC_WD.....	545
Proper Use of Variables	547

MGC_LOCATION_MAP

The default locations for a location map are as follows:

```
./mgc_location_map  
$HOME/mgc/mgc_location_map  
$MGC_HOME/etc/mgc_location_map  
$MGC_HOME/shared/etc/mgc_location_map
```

If the location map does not reside at one of the default locations, you need to set the MGC_LOCATION_MAP environment variable so that the application can read the appropriate location map for the specified project. If the value of MGC_LOCATION_MAP is set to the string “NO_MAP”, the application assumes that you do not intend to use a location map. You can set the value for \$MGC_LOCATION_MAP in a user's login or application startup shell scripts.

Related Topics

MGC_WD	Proper Use of Variables
------------------------	---

MGC_WD

An application uses the value of MGC_WD to set the project working directory for the DDMS. The value of MGC_WD is the location of the project directory using a pathname naming convention that is universal on each system. Setting MGC_WD is important because the DDMS and UNIX pathname resolution mechanisms are not equivalent. UNIX versions of the current working directory differ between shells and may not be interoperable.

Also, if your network uses the NFS automounter to mount remote file systems on demand, the pathname used to get to an object and the pathname returned through the UNIX **pwd** command may not be equivalent. For example:

```
$ cd /net/carson/obj  
$ pwd  
/tmp_mnt/net/carson/obj
```

In the above example, if the mount time on file system *carson* had expired and the file system been dismounted, you could not contact *carson* again through the path */tmp_mnt/net/carson* that had been returned by **pwd**.

The DDMS caches the current working directory in memory and resolves relative pathnames by prepending the current working directory to the start of the relative pathname. To resolve the pathname appropriately and to avoid having temporary mount points stored in Mentor Graphics designs, the DDMS needs to know the current working directory and correct pathname syntax for the project.

If the current working directory is not set through MGC_WD, the DDMS uses UNIX and issues the following warning:

```
//Warning: Your MGC_WD environment variable is not set.  
<pathname> will be used for your working  
directory. Improper design references  
might be created using relative pathnames with  
this working directory. To change your  
working directory, use the menu MGC > Location  
Map > Set Working Directory or the  
$set_working_directory() function.
```

Set MGC_WD in the login script or application startup file for each user. For example, if the project directory is */usr1/proj_x*, enter the following line in the login script or application startup file for Bourne or Korn shell users:

```
MGC_WD=/usr1/proj_x  
export MGC_WD
```

Enter the following line in the login script for the project's C shell users:

```
setenv MGC_WD /usr1/proj_x
```

If a user works on multiple projects simultaneously, you may want to create a shell startup script for each project. Include the following line to synchronize the UNIX and DDMS pathname resolution mechanisms:

```
cd $MGC_WD
```

Related Topics

[MGC_LOCATION_MAP](#)

[Proper Use of Variables](#)

Proper Use of Variables

Using the MGC_LOCATION_MAP and MGC_WD environment variables improperly could cause problems with pathname resolution.

To prevent such problems, adhere to the following guidelines:

- Environment variables corresponding to location map soft prefixes should not be set when you invoke a server process, even though the values of the environment variables are the same as the soft prefixes. Servers read the location map file when they start a job or process. If the location map changes when environment variables are set at invocation time, the processes on the server must be stopped and restarted, since environment variables override location map entries.
- The project team must keep track of environment variables used to reference design data and the corresponding hard pathnames. Conflicting environment variables prevent data sharing, as do environment variables invoked by one project team member and not invoked by another team member. Use the location map to maintain soft prefixes.

You can create shell environment variables from the soft prefixes in a specified map. The **ddms_locenv** shell script reads a location map and, based on the entries found in the location map, creates a file with shell-specific syntax to define environment variables. You can then execute the file created by **ddms_locenv** to define the environment variables in the current shell.

Related Topics

[MGC_LOCATION_MAP](#)

[MGC_WD](#)

Location Map Network Administration Examples

There are three network scenarios for location maps:

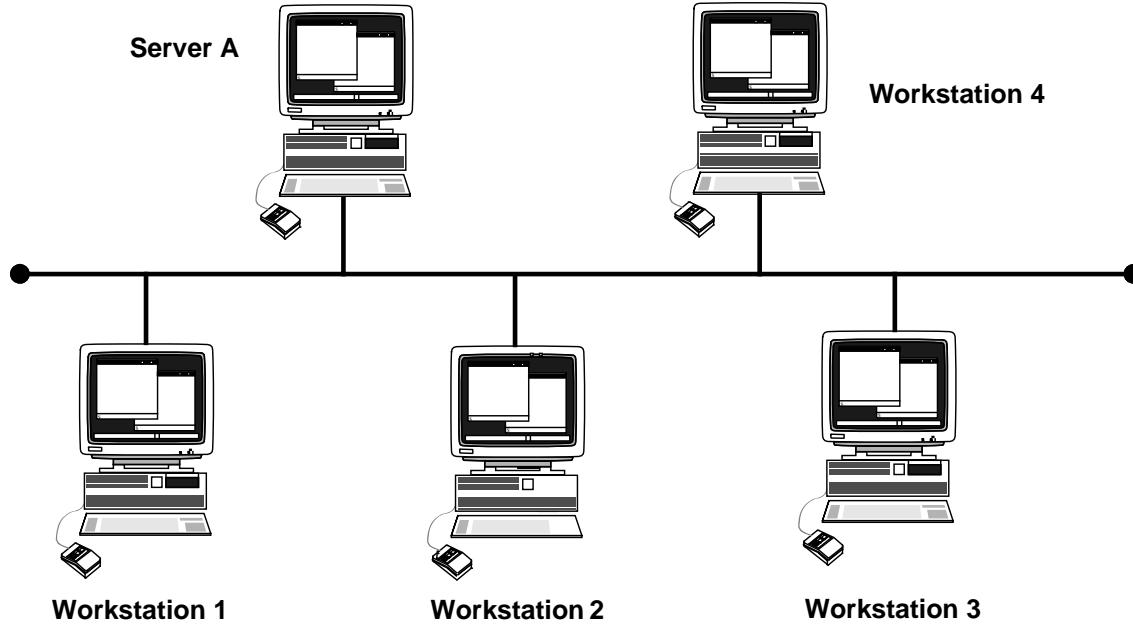
Server Environment	547
Small Network with Direct NFS Mounts	549
Large Networks	549

Server Environment

If project data is stored on one or more file servers, project members probably share a single location map with common path to the information through path names that cross workstation boundaries.

The project members access the project information through NFS mount points on the server. The following shows the network layout for four workstations and a server on an ethernet.

Figure H-4. Location Map Server Environment



All of the client workstations mount the project directory on the server, which contains the design files for Project X located at */projects/project_x*. The client workstations mount the server's */projects* directory at */server_a/proj* and access both the project location map and design projects at that location. The client workstations reference the project location map by using the *MGC_LOCATION_MAP* environment variable, which is set as follows:

```
MGC_LOCATION_MAP=/server_a/proj/project_x/location_map
export MGC_LOCATION_MAP
```

The project uses *gen_lib*, which is located at */usr1/component_libs* on *server_a*. The client workstations mount */usr1* from *server_a* at */server_a/component_libs*. The project location map on *server_a*, which is the same for each client workstation, looks like the following example:

```
MGC_LOCATION_MAP_1
#####
# Project X Project Location Map
# File /projects/project_x/location_map
# Revision 2.2, August 1999
# Administrator: jsmith
#####
# MGC Genlib V8.7_1
$MGC_GENLIB
/server_a/component_libs/mgc_genlib
# Project X Designs
$PROJECT_X
/server_a/proj_x/project_x
```

Related Topics

[Small Network with Direct NFS Mounts](#)[Large Networks](#)

Small Network with Direct NFS Mounts

In sites with small networks (ten or fewer machines), every workstation in a project group may have a mount point to access the other workstations' project data.

For example, workstation_a, workstation_b, and workstation_c comprise a small network of three workstations. If the project team decides to locate project data in */proj/mod_1* on workstation_a, */proj/mod_2* on workstation_b, and */proj/mod_3* on workstation_c, with an ASIC library at */proj/asic_lib* on workstation_a, the entries for the location map are configured as shown in a previous table.

Table H-1. Workstation Location Maps

Workstation A	Workstation B	Workstation C
\$MOD1 <i>/proj/mod1</i>	\$MOD1 <i>/a/mod_1</i>	\$MOD1 <i>/a/mod_1</i>
\$MOD2 <i>/b/mod_2</i>	\$MOD2 <i>/proj/mod_2</i>	\$MOD2 <i>/b/mod_2</i>
\$MOD3 <i>/c/mod_3</i>	\$MOD3 <i>/c/mod_3</i>	\$MOD3 <i>/proj/mod_3</i>
\$ASIC_LIB <i>/proj/asic_lib</i>	\$ASIC_LIB <i>/a/asic_lib</i>	\$ASIC_LIB <i>/a/asic_lib</i>

The mount points for each workstation are:

- workstation_a mounts workstation_b's */proj* directory at */b* and workstation_c's */proj* directory at */c*.
- workstation_b mounts workstation_a's */proj* directory a */a* and workstation_c's */proj* directory at */c*.
- workstation_b mounts workstation_a's */proj* directory a */a* and workstation_b's */proj* directory at */b*.

Related Topics

[Server Environment](#)[Large Networks](#)

Large Networks

A large network requires the system manager to pay close attention to the master location map, with care taken to avoid conflicts and soft prefixes with duplicate hard pathnames.

Large networks are generally composed of combinations of the scenarios described in “Server Environment” and “Small Network with Direct NFS Mountss.”

Related Topics

[Server Environment](#)

[Server Environment](#)

[Small Network with Direct NFS Mounts](#)

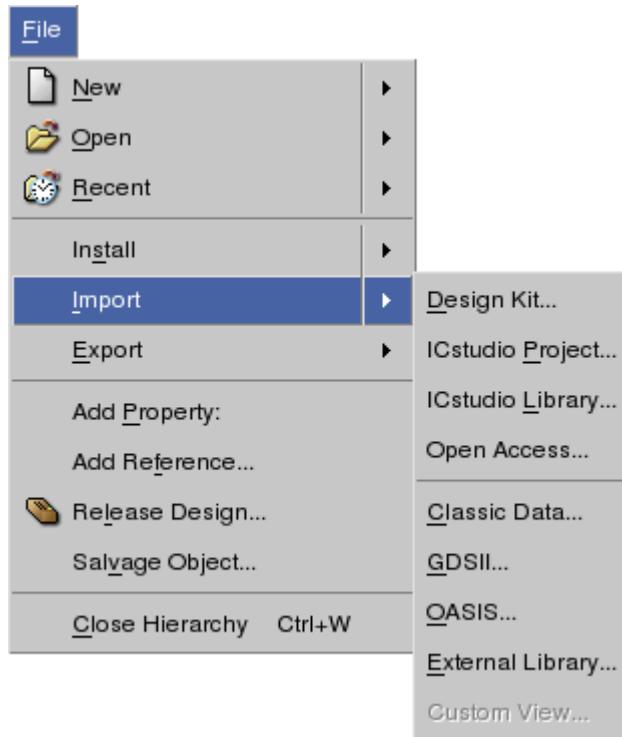
Appendix I

Importing Design Data

You may need to import existing IC Studio and Classic data into the hierarchical components provided by Project Navigator within the Pyxis Project Manager tool.

The import utilities are found under the **File > Import** pulldown menu options, as shown in Figure I-1.

Figure I-1. File > Import Pulldown Menu Options



Note

 In order to use any of these Import options, you need to have read permissions at the source data location and write permission at the destination.

Design data can be migrated or imported in to a project within the project navigator scope of Pyxis Project Manager. You can import a Design Kit, OpenAccess, Classic Data, GDSII, OASIS, an external library, or a custom view. For information on OpenAccess, see the chapter on “[OpenAccess Import/Export](#)” on page 407.

Importing a Technology Design Kit..... [552](#)

Importing an IC Studio Project	553
Importing an IC Studio Library.....	557
Importing Classic Data	559
Importing External Library	562
Importing Custom View	563

Importing a Technology Design Kit

Import a TDK into Project Manager.

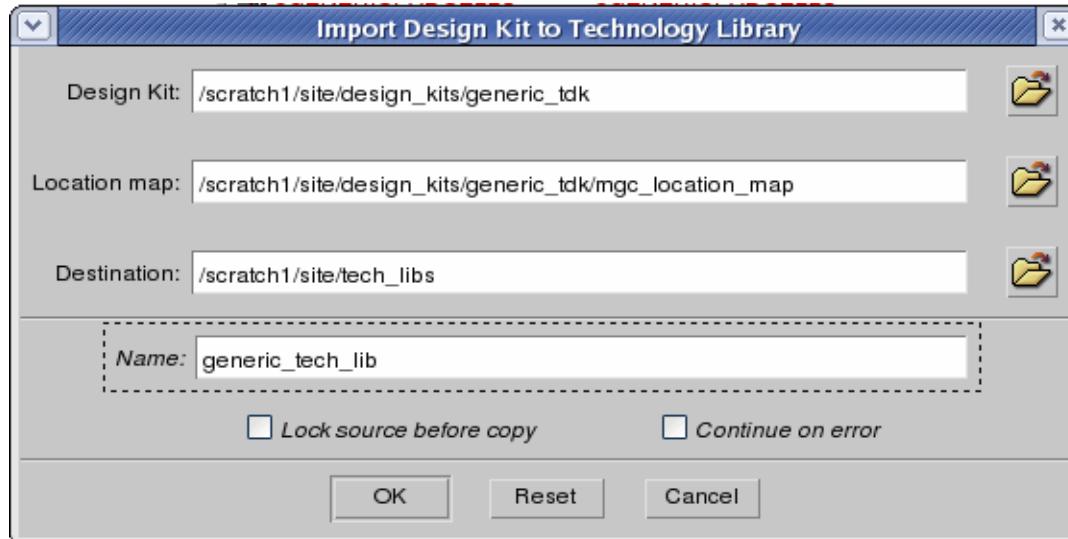
Prerequisites

Before importing any existing project data in to the Project Navigator format, you are required to import the Technology Design Kit (commonly called the TDK) into a technology library.

Procedure

1. Access the pulldown menu option under **File > Import > Design Kit**. This brings up the Import Design Kit to Technology Library dialog box, as shown in [Figure I-2](#).

Figure I-2. Import Design Kit to Technology Library Dialog Box



2. The options have the following meaning:

- Design Kit

Type in or browse to the path for the source Technology Design Kit or TDK.

- Location map

Type in or browse to the path for the location map that is used by the existing TDK. This generally resides directly under the TDK directory.

- Destination

Type in or browse to the directory location of the technology library.

Caution



Do not use environment overrides when importing data.

- Name (optional)

Specify the name of your choice for the technology library.

- Lock source before copy

This option specifies whether or not to lock permissions to the objects under the source Technology Design Kit before performing the copy. This ensures that no other users modify the contents of the source during the copy. By default, this option is not selected.

- Continue on error

If there is an error during the import process, this option directs Pyxis Project Manager to initially log the error in the session monitor, and then continue with the copy process. Additionally, Pyxis Project Manager reports any broken references created in the destination technology library. If an error occurs when you do not enable this option, Pyxis Project Manager logs the error in the session monitor, and then removes any data created at the destination location.

Note



Once the Technology Design Kit is successfully imported into a technology library inside of Project Navigator, you need to create at least one technology configuration within the technology library.

3. Click the OK button to execute the Import Design Kit to Technology Library dialog box.

Related Topics

[Creating a Technology Configuration](#)

Importing an IC Studio Project

There are some advantages to importing your IC Studio project into Pyxis Project Manager's Project Navigator.

- Improved integration

The Project Navigator window of Pyxis Project Manager provides essentially the same library, cell, and view structure that IC Studio offered. However, because Pyxis Project Manager uses the same underlying foundation as Pyxis Schematic and Pyxis Layout, Project Navigator is much better integrated with both these tools in terms of look and feel.

- Improved scripting capabilities

Pyxis Project Manager supports the highly usable Ample scripting language to enable flexibility in both customization and data management scripts.

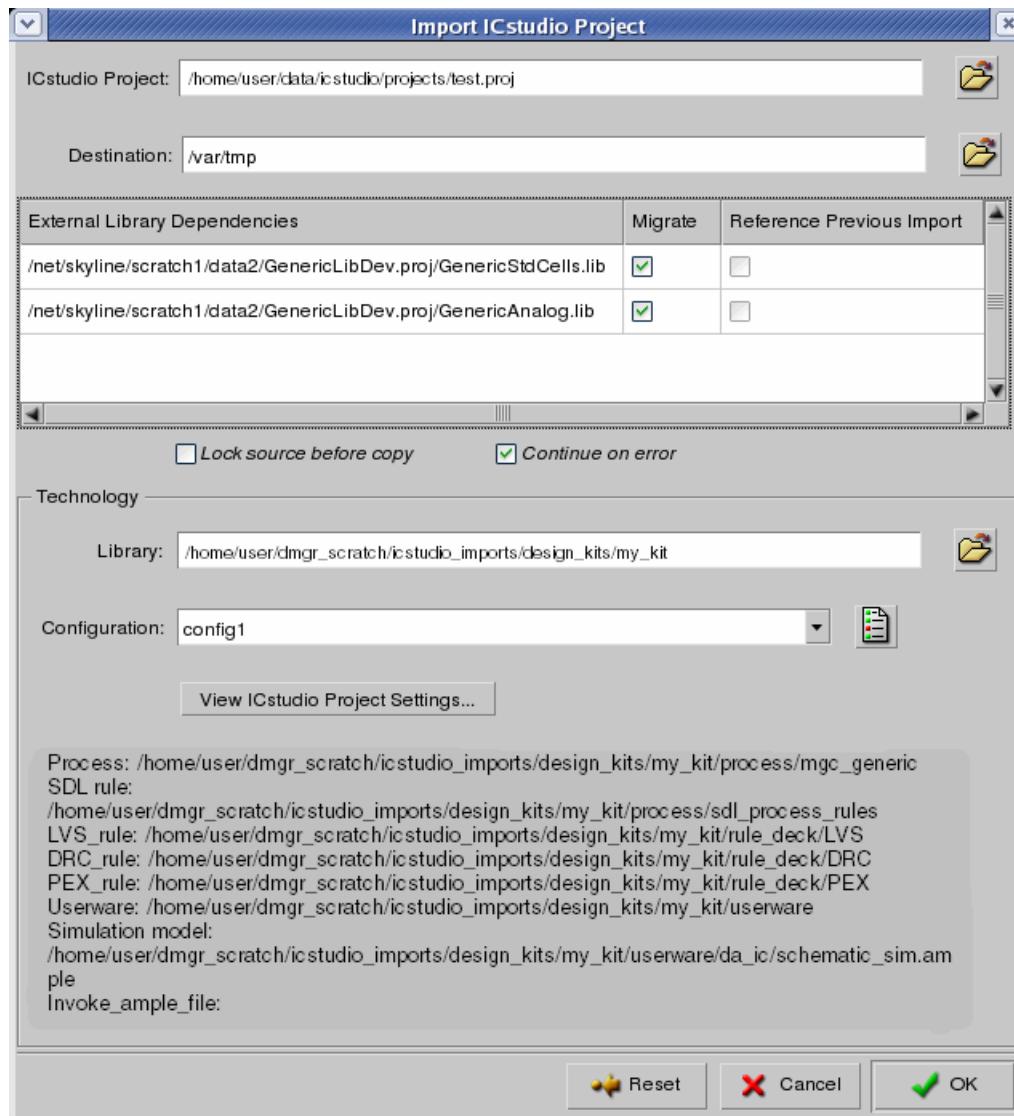
- Improved data management capabilities

Project Manager has improved data management capabilities, including automatic re-referencing for renamed cells, as well as real time updates of MGC_LOCATION_MAP, (which allow Pyxis Project Manager users to add or change libraries while the design data is open).

Procedure

1. Access the pulldown menu option under **File > Import > ICstudio Project**. This brings up the Import ICstudio Project dialog box, as shown in [Figure I-3](#).

Figure I-3. Import ICstudio Project Dialog Box



2. The options have the following meaning:

- **ICstudio Project**

Type in or browse to the path for the source IC Studio Project.

- **Destination**

Type in or browse to the directory location of the Project.

- **External Library Dependencies table**

Once you specify a valid IC Studio Project, the table in this dialog box gets populated with a list of external library dependencies.

If you select Migrate in the External Library Dependencies table, the external libraries that are migrated as part of the import operation are placed as peers to the imported project.

Note

 Pyxis Project Manager does not import the IC Studio data, if there are any broken references. Hence, use the Check/ Fix References feature of IC Studio to fix any broken references before importing the IC Studio data into Pyxis Project Manager.

Pyxis Project Manager chooses destination directory for each external library that is imported. A checkmark in the “Reference Previous Import” column indicates that the software has previously imported that external library.

Also, the source library is not imported, and any references are mapped to the existing external library.

Note

 The working directory for imported design configuration viewpoints is not updated by Pyxis Project Manager. You can correct this by opening the viewpoint in Pyxis Schematic, then using the “Session” palette menu button, and selecting the “Simulator/Viewer” popup menu item. In the dialog box that opens up, update the path in the “Simulator invoked from directory:” field.

- **Migrate**

When you select the Migrate column in the external library dependency table, the external libraries that are migrated as part of the import operation are placed as peers to the imported project. This is mutually exclusive with the “Reference Previous Import” column.

- **Lock source before copy**

This option specifies whether or not to lock permissions to the objects under the IC Studio project before performing the copy. This ensures that no other users modify the contents of the source during the copy. By default, this option is not selected.

- **Continue on error**

If there is an error during the import process, this option directs Pyxis Project Manager to initially log the error in the session monitor, and then continue with the copy process. Additionally, Pyxis Project Manager reports any broken references created in the destination technology library. If an error occurs when you do not enable this option, Pyxis Project Manager logs the error in the session monitor, and then removes any data created at the destination location.

It may be useful to enable this option while closely monitoring the session monitor, so as to address any errors related to the importing of specific design objects (after the import process completes).

- Technology Library

Type in or browse to the technology library that is attached to the Project. The technology library must be a Pyxis Project Manager library, not an ICstudio technology library.

Caution

 Do not use environment overrides when importing data. For example, if the softpath of the technology library (such as \$MGC DESIGN KIT) is set through an environment variable, which points to some path other than the path defined in the mgc_tech_location_map, then when you import an ICstudio project, the GUI does not use the softpath in the "library" item, and instead uses a hard path. The resulting data then contains a hard path in the project's attribute file. Also, it is not possible to override it later because it uses a hard path rather than a soft path for the reference to the technology library.

- Technology Configuration

Specify the technology configuration of the technology library that is attached to the project.

- View ICstudio Project Settings

To assist with the import process, the Import ICstudio Project dialog box includes the Settings button.

When you click the View ICstudio Project Settings button, it brings up a read-only View Project Settings window and provides a description of the configuration details of the source IC Studio project.

Related Topics

[Importing an IC Studio Library](#)

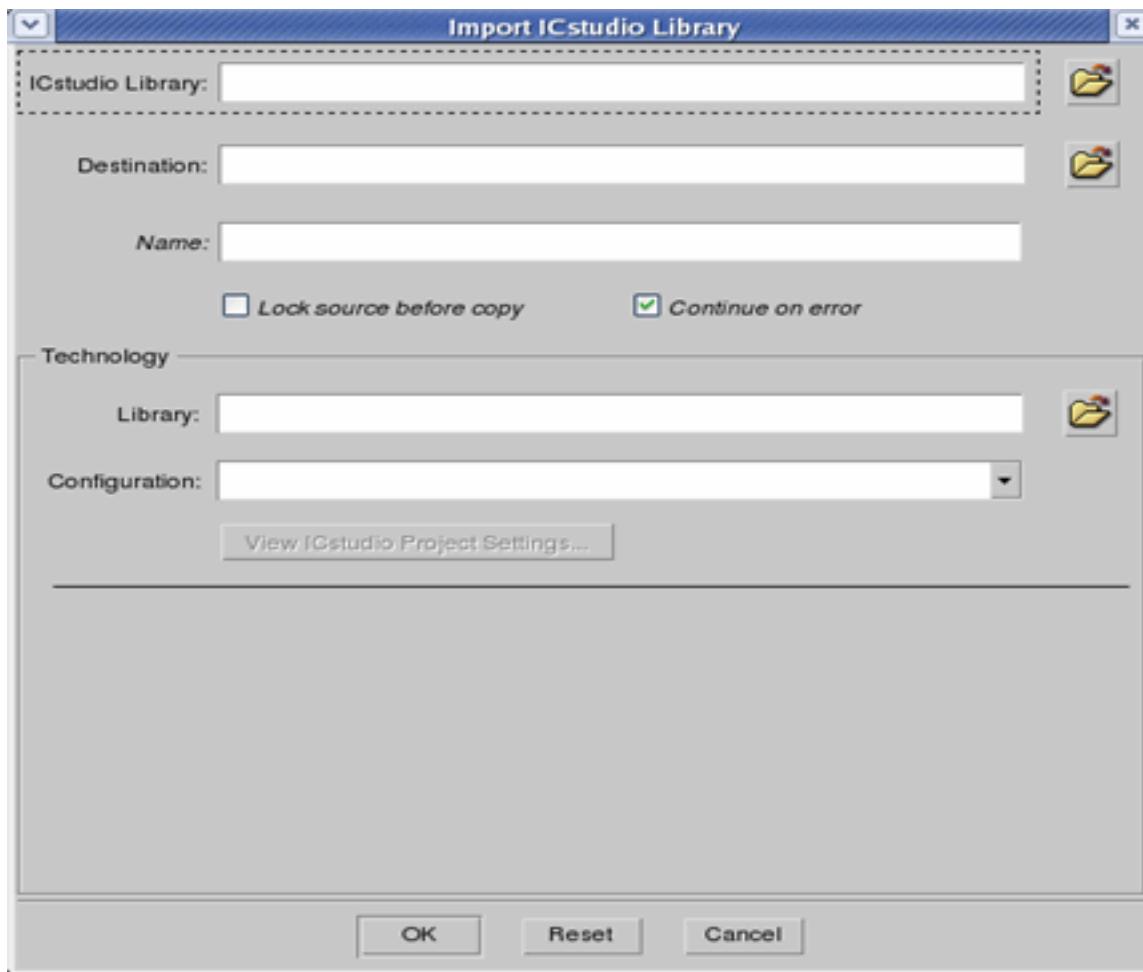
Importing an IC Studio Library

Import an IC Studio library into an external library within the Project Navigator.

Procedure

1. Access the pulldown menu option under **File > Import > IC Studio Library**. This brings up the Import IC Studio Library dialog box, as shown in [Figure I-4](#).

Figure I-4. Import IC Studio Library Dialog Box



2. The options have the following meaning:

- IC Studio Library

Type in or browse to the path for the source IC Studio Library.

- Destination

Type in or browse to the directory location of the external library.

- Name (optional)

Specify the name of your choice for the external library.

- Lock source before copy

This option specifies whether or not to lock permissions to the objects under the IC Studio project before performing the copy. This ensures that no other users modify the contents of the source during the copy. By default, this option is not selected.

- Continue on error

If there is an error during the import process, this option directs Pyxis Project Manager to initially log the error in the session monitor, and then continue with the copy process. Additionally, Pyxis Project Manager reports any broken references created in the destination technology library. If an error occurs when you do not enable this option, Pyxis Project Manager logs the error in the session monitor, and then removes any data created at the destination location.

It may be useful to enable this option while closely monitoring the session monitor, so as to address any errors related to the importing of specific design objects (after the import process completes).

- Technology Library

Type in or browse to the technology library that is attached to the imported project.

- Technology Configuration

Specify the technology configuration of the technology library that is attached to the imported project. When selected, the technology settings for the specified technology configuration appears in the technology list below it.

3. Click OK to execute the Import IC Studio Library dialog box.

Related Topics

[Importing an IC Studio Project](#)

Importing Classic Data

There are some advantages to importing your Classic data into Pyxis Project Manager's Project Navigator.

- Standard Organization of Project Data

In the Classic flow, design groups had the flexibility to organize project data into a structure that best suited their needs. While this enabled them to create the optimal structure for their specifications, it is generally more difficult for work to be shared among multiple sites. Project Navigator provides standard organization of project data by introducing new hierarchical design objects.

Note



The working directory for imported design configuration viewpoints is not updated by Pyxis Project Manager. You can correct this by opening the viewpoint in Pyxis Schematic, then using the "Session" palette menu button, and selecting the "Simulator/Viewer" popup menu item. In the dialog box that opens up, update the path in the "Simulator invoked from directory:" field.

The following Hierarchical Design Objects can be found in Project Navigator:

- Project — A container of design data that is under development. This can be organized in libraries and categories.
- External Library — An independent container of design data that can be referenced by multiple projects.
- Technology Library — A container of technology specific data that is referenced by projects and external libraries.
- Revision Control Interface

A revision control interface enables project data organized in the Project Navigator format to be placed under revision control. This helps multiple users perform concurrent design.

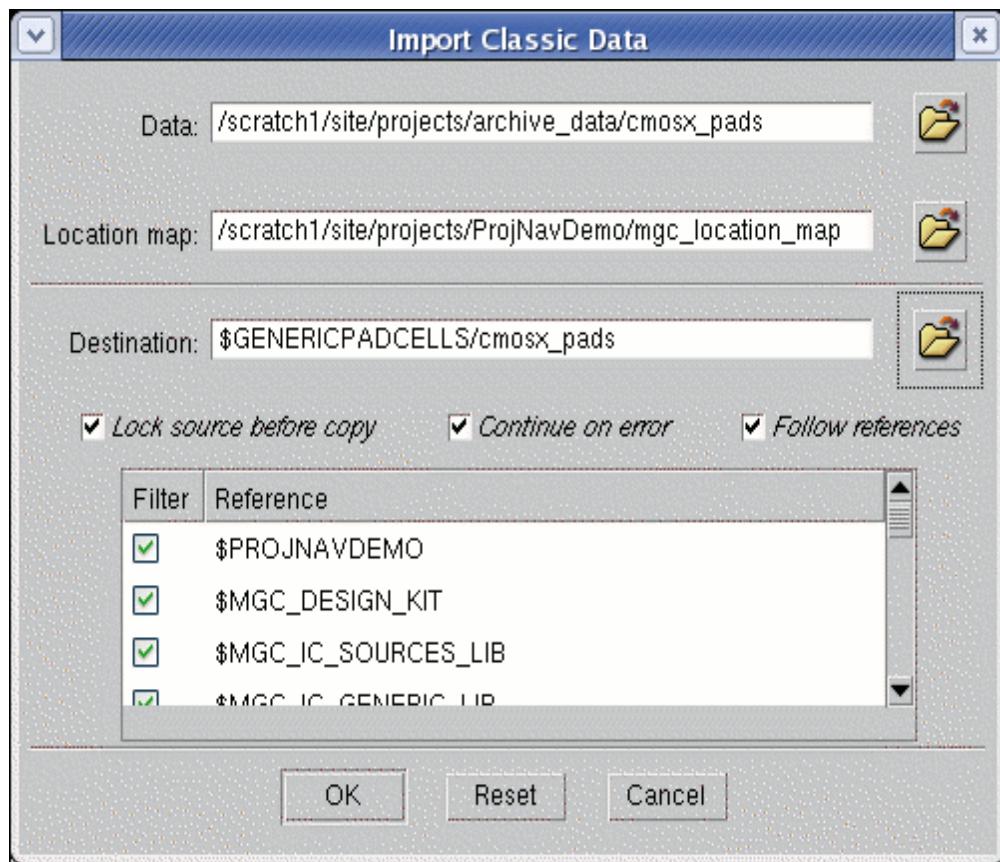
Caution



Do not use environment overrides when importing data.

Procedure

1. Access the pulldown menu option under **File > Import > Classic Data**. This brings up the Import Classic Data dialog box, as shown in [Figure I-5](#).

Figure I-5. Import Classic Data Dialog Box

2. The options have the following meaning:

- Data

Type in or browse to the path for the location of the Classic Data.

- Location map

Type in or browse to the path for the location map that is used to find the references in the Classic Data.

- Destination

Type in or browse to the existing Project Navigator container, into which the Classic Data is imported. The container could be a library, an external library, or a category under either a library or an external library.

- Lock source before copy

This option specifies whether or not to lock permissions to the objects under the Classic Data before performing the copy. This ensures that no other users modify the contents of the source during the copy. By default, this option is not selected.

- Continue on error

If there is an error during the import process, this option directs Pyxis Project Manager to initially log the error in the session monitor, and then continue with the copy process. Additionally, Pyxis Project Manager reports any broken references created in the destination technology library. It may be useful to enable this option while closely monitoring the session monitor, so as to address any errors related to the importing of specific design objects (after the import process completes).

- Follow references

Select this option if you want to import all the data objects that are referenced by the Classic Data into the Destination directory.

- Filter/Reference table

When the “Follow references” option is enabled, this table allows you to select which references to skip during the import process.

3. Click OK to execute the Import Classic Data dialog box.

Related Topics

[Revision Control](#)

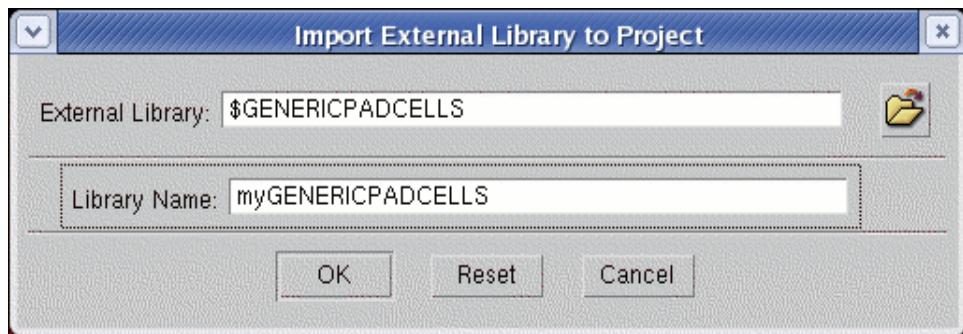
Importing External Library

Import an IC Studio external library into the active project within the Project Navigator scope.

Procedure

1. Access the pulldown menu option under **File > Import > External Library**. This brings up the Import External Library to Project dialog box, as shown in [Figure I-6](#).

Figure I-6. Import External Library Dialog Box



2. The options have the following meaning:

- External Library
Type in or browse to the path for the source IC Studio external library.
 - Library Name
3. Click OK to execute the dialog box.

Related Topics

[Importing an IC Studio Library](#)

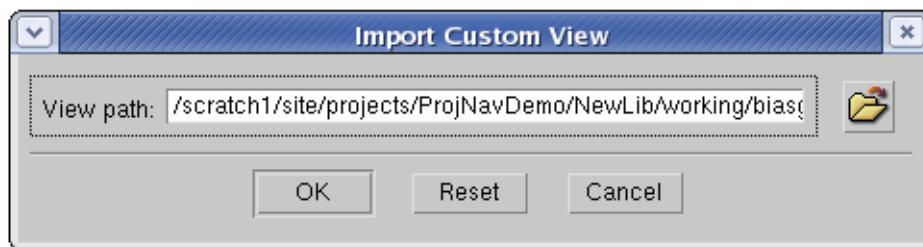
Importing Custom View

Import a Custom View into the selected cell within the Project Navigator scope.

Procedure

1. Access the pulldown menu option under **File > Import > Custom View**. This brings up the Import Custom View dialog box, as shown in [Figure I-7](#).

Figure I-7. Import Custom View Dialog Box



2. In the View Path field, type in or browse to the path where the view is located.



Note

Before the imported view becomes accessible within Pyxis Project Manager, you must confirm that the Custom View Type has been registered with the system.

3. Click OK to execute the Import Custom View dialog box.

Appendix J

EDIF Converter Quick Start Guide

You may need to convert EDIF to Pyxis Schematic. A Pyxis Schematic license is required to run the EDIF converter.

EDIF Converter	565
Command File	569
NCF Generation	579

EDIF Converter

The EDIF converter only works with Pyxis Project Manager projects.

Acronyms	565
Usage.....	565
Main Functionality	567
Features Provided	567
About Schematic Language Generation in the NCF.....	568
Assumptions	568

Acronyms

Following are the descriptions of some acronyms that are used in this chapter:

- CDF - Component Description Format
- NCF - Netlist Control File

Related Topics

[Usage](#)

[Main Functionality](#)

Usage

Run the conversion tool and specify arguments.

Procedure

1. Type the command line using the following syntax:

```
$MGC_HOME/bin/ converter schematicEdif... [-cmd cmd_file...]  
[-n netlist_edif_file...] [-cdf cdf_file...] [-v] [-i] -p project/techlib_path
```

The command syntax entries have the following meaning:

[]	Means optional
...	Means one or more occurrences
converter	Tool name
<schematicEdif>	Name of the main input EDIF file(s)
-cmd <cmd_file>	Entering one or more command files
-n <netlist_edif_file>	Entering one or more netlistEdif files
-cdf <cdf_file>	Entering one or more CDF files
-v	Work in verbose mode, default is silent
-i	Work in interactive mode, default is batch
-p <project/techlib_path>	The output Pyxis Project Manager project (or technology library) path, which should be created before running the conversion command

2. The following usage note displays upon invoking the tool with no arguments:

```
$MGC_HOME/bin/ converter  
converter: No input specified
```

EDIF Importer Usage:

```
converter <edif_file>... [-cdf <cdf_file>...] [-n <netlist_edif_file> ...] [OPTIONS]  
-p <project/techlib_path>
```

OA Importer Usage:

```
converter -oa <oa_lib_def_file> [OPTIONS] -p <project/techlib_path>
```

The usage note has the following meaning for the options:

OPTIONS:

-cmd <cmd_file>...	List of command files
-v, -verbose	Set verbose mode on
-i, -interactive	Set interactive mode on

Related Topics

[Main Functionality](#)

[Features Provided](#)

Main Functionality

This product should take the inputs listed in the above Usage section, given that:

- Input EDIF is the correct EDIF 200 Schematic format, which is exported using Cadence Virtuoso "EDIF 200 out" tool.
- The output is schematics, symbols, and NCF files for the Pyxis Project Manager project that you specified.

Note



The Pyxis Custom Design flow only supports the EDIF version created by Virtuoso schematic.

Related Topics

[Features Provided](#)

Features Provided

The Pyxis EDIF converter has the following features:

- The converter parses the provided inputs:
 - a. The input Schematic EDIF: For getting the main schematic data.
 - b. The command file(s) (optional): For getting user commands and options.
 - c. The CDF file(s) (optional): For getting the graphical data described in the CDF files.
 - d. The Netlist EDIF file(s) (optional): For getting complete information about inherited connections if any exist in the design.
- The converter generates symbols and schematics according to the graphical data provided in both Schematic EDIF file(s) and CDF file(s).
- The converter also generates an NCF from CDF inputs.
- The converter generates inherited connections' data correctly and completely, if they exist, and if the Netlist EDIF file is provided.
- You can define different options using the command file. The options include renaming cells, replacing cells, and editing properties. A complete list of these options is described in detail in the Command file section.

Related Topics

[Command File](#)

About Schematic Language Generation in the NCF

The logic behind generating the SCHEMATIC language in the NCF file for a device that has a schematic defined for it is as follows:

1. If there is a valid language (Eldo or HSPICE) defined in the CDF, then two conditions must be satisfied to have a valid language:
 - The namePrefix is defined and has a non-null value.
 - There is a termOrder defined for that language.In this case, the NCF contains the language EldoSPICE or HSPICE only and no SCHEMATIC language is output.
2. If there is a valid Eldo or HSPICE language defined in the CDF but the componentName property of that language is "subcircuit", then the NCF contains the SCHEMATIC language only given that there is no Eldo or HSPICE language that is valid to be put in the NCF file.
3. If there is no valid Eldo or HSPICE language defined in the CDF, then the SCHEMATIC language only outputs in the NCF file.

Related Topics

[Assumptions](#)

Assumptions

The following assumptions are made with respect to the EDIF Converter's functionality:

- Input EDIF is the correct EDIF 200 Schematic format exported using Cadence Virtuoso "EDIF 200 Out" tool.
- The EDIF Converter tool is not responsible for generating simulation models or layout data.
- It is your responsibility to validate the conversion. Connectivity can be checked using Calibre LVS to perform a netlist vs. netlist comparison after the conversion.

Related Topics

[EDIF Converter](#)

Command File

Following are the features of the command file:

- The purpose of the command file is to enable you to specify more options and directives.
- All commands are case insensitive.
- Add comments in the command file by using “//”.
- There are three scope identifiers: “-cat”, “-lib”, and “-cell”. The intersection between these identifiers defines the scope on which the command operates.

For example, consider the following command:

```
renameNet (input output) -cell a b c -lib x y;
```

The command is applied on cells **a**, **b**, and **c** in libraries **x** and **y** only.

- The general syntax for the command is:

```
command <input> [-option1 <argument>] [-option2 <argument>]
```

The entries for the command syntax have the following meaning:

[]	Means optional, zero or one times
[...]	Zero or more times
...	One or more times, without square braces
	Or
0	Grouping. In the following example, the whole pair can be mentioned one or more times: (<input1><input2>)...

These parentheses must exist while calling the command. For example, to rename cell "x" to be "y", the syntax is:

```
renameCell (x y) -lib lib1
```

;
Any command is terminated by ";".

- Commands are executed sequentially.
- Additional commands in the included file are executed immediately after the execution of the main command file. That means, newly added commands in the included file are appended to the main file. If there is more than one included file, each is appended according to its order in the main command file. Recursive includes are not allowed.

Command Syntax	570
Sample Command Files	576

Command Syntax

The following are the types of commands available with the EDIF Converter:

Pin Commands

```
renamePort (<oldName> <newName>) [-cell <cellName>] [-cat <catName>] [-lib <libName>];  
  
replacePort (<oldPortCellPath> <newPortCellPath>) [-cell <cellName>] [-cat <catName>] [-lib <libName>];
```

Text and Shape Notes Commands

```
deleteNote <"text"> [-cell <cellName>] [-cat <catName>] [-lib <libName>] [-e];
```

Sheet Border Commands

```
deleteBorder [-cell <cellName>] [-cat <catName>] [-lib <libName>];
```

Property Commands

```

addProperty <propertyName> | (<propertyName> <propertyValue>)
[-cell <cellName>] [-cat <catName>] [-lib <libName>]
[-type <string | number | triplet | expression >]
[-height=nnn]
[-symbolVisib <ON | OFF>]
[-just<LB|LC|LT|CB|CC|CT|RB|RC| RT >]
[-location (x,y) ]
[-rotation<0|90|180|270>]
[-stability<FIXED| PROTECTED|NONREMOVABLE|VARIABLE>]
[-SPEFormLabel <ParPrompt>]
[-SPEFormDefault <ParDef>]
[-SPEUserVisible <ParVisibility>]
[-SPEReadOnly <ParEditable>]
[-SPEUserChangeCbk <ParCallback>]
[-SPEChoices <ParChoicesList>]
[-SPEController <Controller>]
[-SPERangeLow <ParRangeLowCB>]
[-SPERangeHigh <ParRangeHighCB>]
[-SPEHelp<ParHelpMsg>];

editProperty [<propertyName> | (<propertyName > <propertyValue>) ...]
[-cell <cellName>] [-cat <catName>] [-lib <libName>]
[-type <string | number | triplet | expression >]
[-height=nnn]
[-symbolVisib <ON | OFF>]
[-just<LB|LC|LT|CB|CC|CT|RB|RC| RT >]
[-rotation<0|90|180|270>]
[-stability<FIXED| PROTECTED|NONREMOVABLE|VARIABLE>]
[-location (x,y) ]
[-SPEFormLabel <ParPrompt>]
[-SPEFormDefault <ParDef>]
[-SPEUserVisible <ParVisibility>]
[-SPEReadOnly <ParEditable>]
[-SPEUserChangeCbk <ParCallback>]
[-SPEChoices <ParChoicesList>]
[-SPEController <Controller>]
[-SPERangeLow <ParRangeLowCB>]
[-SPERangeHigh <ParRangeHighCB>]
[-SPEHelp<ParHelpMsg>];

renameProperty (<oldname> <newName>) [-cell <cellName>] [-cat <catName>]
[-lib <libName>];

deleteProperty <propertyName> [-cell <cellName> ][ -lib <libName>] [-cat
<catName>] ;

calculateProp (statement) [-cell <CellName>] [-lib <libName>] [-cat
<CatName>] [-instOf <CellName>];

```

Schematic Sheet Commands

```

deleteSheet <sheet Name> [-cell <cellName>] [-lib <libName>] [-cat
<catName>];

```

Cell Commands

```
renameCell (<oldName> <newName>) [ -cat <catName>] [-lib <libName>];  
replaceCell (<oldPath> <newPath>) [ -cat <catName>] [-lib <libName>];  
deleteCell <cellName> [-cat <catName>] [-lib <libName>];  
ScaleConvertedCells <ScaleValue>;
```

The syntax for the Cell commands has the following meaning:

- **ScaleValue** — This value can be less than 1 denoting a down-scaling or greater than 1 denoting an up-scaling.
For example, 0.75 represents a 75% scaling of all symbols and schematics and 1.2 represents a 120% scaling of all symbols and schematics.

Lib Commands

```
renamelib (<oldName> <newName>);  
deletelib <libName>;  
replaceLib (edifLibName newLibPath) -map cellMapOptions;
```

The syntax for the Lib commands has the following meaning:

- **newLibPath** — This should be the path of the new library that includes the cells under its folder. If one or more of the cells under **edifLibName** do not exist under **newLibPath**, the **edifLibName** is created to include only the missing cells, unless **-map** option is used.
- **cellMapOptions** — This can be one of the following three options:
 - a. **(edifCellName newCellName)** — Maps cells with different names in the EDIF library and the new library.
 - b. **(anotherEdifLibName/edifCellName newCellName)** — Maps cells from another library to a cell in the new library.
 - c. **(edifCellName property (value newCellName) (anotherValue anotherNewCellName))** — Performs conditional mapping in the following pattern:
 - Maps instances of the cell **edifCellName** according to the value of the property **property**.
 - For value **value**, instantiates **newCellName** from the linked library.
 - For value **anotherValue**, instantiates from **anotherNewCellName** from the linked library.
 - You can then add multiple **value/newCellName** pairs as needed for the same **edifCellName** and **property**.

- d. **(edifCellName newCellName/symbolName)** — Maps one cell to another with a specified symbol view name.

General Commands

```
Include <file>;  
  
additionalEDIF <path>;  
  
additionalCDF<path>;  
  
additionalNetlistEDIF<path>;  
  
DefineCategory <categoryName> -lib <LibName> [-cell <cellName>];  
  
RemoveEllipseComments;
```

The syntax for the General Commands has the following meaning:

- **RemoveEllipseComments:** Disables or skips the adding of all ellipse comments.

NCF Commands

```
netlistPrefix <prefix> [-cell <cellName>] [-cat <catName>] [-lib <libName>] [-entry <ELDO|SPICE|HSPICE >];  
  
netlistParam <param> [-cell <cellName>] [-cat <catName>] [-lib <libName>] [-entry <ELDO|SPICE|HSPICE >];  
  
addNetlistParam <param> [-cell <cellName>] [-cat <catName>] [-lib <libName>] [-entry <ELDO|SPICE|HSPICE >];  
  
delNetlistParam <param> [-cell <cellName>] [-cat <catName>] [-lib <libName>] [-entry <ELDO|SPICE|HSPICE >];  
  
mapNetlistParam (<sParam><dParam>) [-cell <cellName>] [-cat <catName>] [-lib <libName>] [-entry <ELDO|SPICE|HSPICE >];  
  
addNetlistParamMapping (<sParam> <dParam>) [-cell <cellName>] [-cat <catName>] [-lib <libName>] [-entry <ELDO|SPICE|HSPICE >];  
  
delNetlistParamMapping <sParam> [-cell <cellName>] [-cat <catName>] [-lib <libName>] [-entry <ELDO|SPICE|HSPICE >];  
  
netlistTerm <term> [-cell <cellName>] [-cat <catName>] [-lib <libName>] [-entry <ELDO|SPICE|HSPICE >];  
  
addNetlistTerm <term> [-cell <cellName>] [-cat <catName>] [-lib <libName>] [-entry <ELDO|SPICE|HSPICE >];  
  
delNetlistTerm <term> [-cell <cellName>] [-cat <catName>] [-lib <libName>] [-entry <ELDO|SPICE|HSPICE >];  
  
mapNetlistTerm (<sTerm><dTerm>) [-cell <cellName>] [-cat <catName>] [-lib <libName>] [-entry <ELDO|SPICE|HSPICE >];  
  
addNetlistTermMapping (<sTerm><dTerm>) [-cell <cellName>] [-cat <catName>] [-lib <libName>] [-entry <ELDO|SPICE|HSPICE >];  
  
delNetlistTermMapping <sTerm> [-cell <cellName>] [-cat <catName>] [-lib <libName>] [-entry <ELDO|SPICE|HSPICE >];  
  
NcfSearchL <searchName> [-entry <ELDO|SPICE|HSPICE >];  
  
netlistControlParam <param> [-cell <cellName>] [-cat <catName>] [-lib <libName>] [-entry <ELDO|SPICE|HSPICE>];
```

instOf switch

An additional switch is added with the name "-instof" to force the command to execute only on instances from the specified arguments "cellNames" in the switch. The syntax is:

```
renameProperty (oldProp newProp) -instof cellName ;
```

In the following example, for every instance from cell **nmos**, property **x** is renamed to **y**.

```
RenameProperty (x y) -instof nmos;
```

The tool has a convention that the scope identifiers switches (“-lib”, “-cell”, or “-cat”) must be the first switches appearing in any command (if available), and hence the location of the “-instof” switch is added so as to appear directly after the scope identifiers. In the following example, note that the arguments coming after the “-instof” switch can be either “cellname” or “libname/cellname”

```
editproperty propertyname -lib libraryname -instof cellname;
deleteproperty propertyname -instof libname/cellname1 cellname2;
renameproperty (oldpropname newpropname) -instof libname/cellname;
```

resizeInst command

Use this command to resize certain instances in the schematic. The resize factor is one of the following values: 0.25, 0.5, 0, 2, 4. Other values prompt a warning and the command is ignored. The scope identifiers are optional switches to specify certain instances to be affected by the command. These switches are grouped together (using AND) so that if they appear together, then the intersection of the rules is applied.

The syntax is:

```
resizeInst <resizeFactor> [scopeIdentifiers];
```

The switches are:

- **-ofLib <libnames>**

Identifies one or more (space separated) library names where all instances of symbols from these libraries are resized.

- **-ofCell <cellnames>**

Identifies one or more (space separated) cell names where all instances of symbols from these cells are resized.

- **-ofView <viewnames>**

Identifies one ore more (space separated) view names where all instances of symbols from these views are resized.

Another set of scope identifiers can be used to specify which schematic is affected by the resizing command. There switches are also grouped together (using AND):

- **-lib <libnames>**

Identifies one ore more (space separated) library names where all instances in any schematic in these libraries are resized.

- ***-cell <cellnames>***

Identifies one or more (space separated) cell names where all instances in any schematic in these cells are resized.

- ***-cat <categorynames>***

Identifies one or more (space separated) category names where all instances in any schematic in these categories are resized.

The categories are defined using the define category command shown above under [General Commands](#).

Examples:

```
//Doubles the size of all instances in all schematics.  
resizeInst 2;  
  
//Doubles the size of all instances in all schematics under cell cell1.  
resizeInst 2 -cell cell1;  
  
//Doubles the size of all instances of cell cell2 instantiated in any  
schematic.  
resizeInst2 -ofCell cell2;  
  
//Doubles the size of only instances of cell cell3 instantiated in  
schematics of cell4.  
resizeInst2 -cell cell4 -ofCell cell3;
```

Related Topics

[Sample Command Files](#)

[General Commands](#)

Sample Command Files

These sample command files are provided for your reference.

1. The following example is a sample of a generic command file that can be used for most of the design conversion runs because it removes some properties that usually exist in the EDIF files but do not have corresponding features in Pyxis Schematic:

```

renameProperty (instNamePrefix element);
deleteProperty drawDottedGridOn;
deleteProperty fontHeight;
deleteProperty gridMultiple;
deleteProperty gridSpacing;
deleteProperty hnlVerilogFormatInst;
deleteProperty instance#;
deleteProperty instancesLastChanged;
deleteProperty instViewList;
deleteProperty interfaceLastChanged;
deleteProperty lastSchematicExtraction;
deleteProperty lxIgnoredParamsForCAS;
deleteProperty net#;
deleteProperty pathName;
deleteProperty pathWidth;
deleteProperty pin#;
deleteProperty schGeometryLastRecorded;
deleteProperty schGeometryVersion;
deleteProperty schType;
deleteProperty schXtrVersion;
deleteProperty sheetSize;
deleteProperty startLevel;
deleteProperty stopLevel;
deleteProperty slotId;
deleteProperty schType;
deleteProperty vendorName;
deleteProperty xSnapSpacing;
deleteProperty ySnapSpacing;
deleteProperty zonesWhereUsed;
deleteNote ">~";

```

2. To map symbols from the imported database to existing MGC symbols, here are the commands that should be used.

```

replaceLib (basic $MGC_IC_GENERIC_LIB) -map (ipin portin) -map (opin
portout) -map (iopin portbi) -map (noConn noConn) -map (vdd_inherit
vdd_inherit) -map (vss_inherit vss_inherit) -map (vcc_inherit
vcc_inherit) -map (gnd_inherit gnd_inherit);

replaceLib (analogLib $MGC_OA_CONVERSION_LIB) -map (cap cap2) -map
(cccs cccs_lin) -map (ccvs ccvs_lin) -map (diode diode2) -map (gnd
gnd) -map (iam iam) -map (idc idc) -map (iexp iexp) -map (ind ind2)
-map (ipulse ipulse) -map (ipwl ipwl) -map (isffm isffm) -map (isin
isin) -map (nmos nmos3) -map (nmos4 nmos4) -map (npn npn3) -map
(pmos pmos3) -map (pmos4 pmos4) -map (pnp pnp3) -map (res res2) -map
(vam vam) -map (vccs vccs_lin) -map (vcvs vcvs_lin) -map (vdc vdc)
-map (vdd vdd) -map (vexp vexp) -map (vpulse vpulse) -map (vpwl
vpwl) -map (vsffm vsffm) -map (vsin vsin) -map (xfmr xfmr4);

```

3. The following example describes the contents of a command file that is used for linking the used kit, and which makes use of a netlist EDIF file in addition.

```

replaceLib (foundry13 $SCRATCH/my_proj/kits/FOUNDRY13/Development);
additionalNetlistEDIF
$SCRATCH/my_proj/phy_macro/edifs/phy_macro_1_AA.netlist.edif;

```

The kit that exists in the path `$SCRATCH/my_proj/kits/FOUNDRY13/Development` replaces the kit `foundry13` in the original design.

The `phy_macro_1_AA.netlist.edif` file is the name of the netlist EDIF file that is used. The netlist EDIF file is a file that can be exported together with the main schematic EDIF file, and carries some electrical connectivity information that may be used by the converter if the original design contained inherited connections.

The netlist EDIF file is an optional input that can be specified using the `-n` switch in the command line or through the command file as illustrated in the example.

4. The following example describes a command file for including other command files. A main command file can be specified in the command line using the `-cmd` switch and additional command files can be specified from inside the main command file as shown below.

```
include $SCRATCH/my_proj/phy_macro/cmds/deleteUselessData.cmd;
include $SCRATCH/my_proj/phy_macro/cmds/foundry13Mapping.cmd;
```

Related Topics

[Command Syntax](#)

NCF Generation

The generated NCF language entries that do not have a corresponding simInfo in the CDF are generated with a NULL_ENTRY.

This entry contains one line with the new NCF syntax, “NL_ACTION OPTIONAL=TRUE”. This line is used to disable netlisting for this language.

Example

This is an example for a device that does not have an auCdl nor auLvs simInfo in the CDF. LVS NCF language is generated with a NULL_ENTRY and is used to disable netlisting for this NCF language.

```
LANGUAGE LVS
  DEF_ENTRY NULL_ENTRY
  ENTRY_NAME NULL_ENTRY
    NL_ACTION OPTIONAL=TRUE
  ENDENTRY
ENDLANGUAGE
```

This device does not appear in the netlist when using the LVS netlister. In case corresponding CDF simInfo is present, the NCF language is generated as usual.

Related Topics

[EDIF Converter](#)

[Pyxis Schematic User's Manual](#)

Glossary

absolute pathname

An absolute pathname is a pathname that originates at the root directory. An absolute pathname begins with either a slash (/) or with a dollar sign (\$), and specifies the full pathname for the specified object.

attribute file

An attribute file is a special file contained within a design object that is used to store all attribute information about the design object. The information includes object type, identity, user-defined properties, and design object references. An attribute file is identified by its .attr suffix. *See also “[Metadata Files](#)” on page 513.*

basic container

A basic container is a container design object that has only one fileset member, a directory. *See also “[Design Object Types](#)” on page 519.*

build rules

The build rules tell the Pyxis Project Manager which design objects to include in and which design to exclude from a configuration during a build. You can specify build rules for each primary entry in a configuration. If you do not specify the build rules, the primary entry inherits the default build rules.

configuration

A configuration is a collection of design objects.

configuration entry

A configuration entry is a pointer to a single version of a design object; the design object is part of a configuration.

configuration object

A configuration object is a special type of design object that specifies how a configuration is built. A configuration object is versioned, it references primary and secondary design objects, and it records build rules for each primary design object. *See also “[Opening a New Configuration](#)” on page 239 in Chapter 3.*

configuration window

A configuration window is a bounded area bordered by a rectangular box within which complex groupings of related design objects are collected and operated on as a single unit. *See also “[Opening a New Configuration](#)” on page 239 in Chapter 3.*

container

A container is a design object whose fileset includes one or more directories. Containers can contain other design objects, and, unlike a directory, they have a special, unique icon that indicates their type.

containment hierarchy

The containment hierarchy is the organization of design objects held in containers throughout the file system. In the context of hierarchy, you can think of a container as simply a directory.

contents mode

When the contents of a directory are displayed in a navigator window, the navigator window is in contents mode. The navigator window is always in contents mode, unless you have just explored the references of a design object. In contents mode, the navigator title bar displays the absolute pathname of the directory whose contents are displayed in the navigator window.

Design Data Management System (DDMS)

The Design Data Management System is responsible for storing, managing, and retrieving persistent data objects requested by applications. The DDMS is a set of classes and their member functions supporting tool and data encapsulation, data model definition and storage, and design data management. The fundamental concepts supported by these functions include persistence, typed objects, object identification, versioning, configurations, transparent referencing of persistent objects, and concurrency control.

default build rules

The default build rules are the rules used to build a configuration if you do not specify build rules for a primary entry. The default build rules include all design objects in both the containment hierarchy and reference network. *See also “[Specifying the Build Rules](#)” on page 242 in Chapter 3.*

design data configuration management

Configuration management is the process of managing configurations. Configuration management includes copying and releasing designs. *See also “[Opening a New Configuration](#)” on page 239, and “[Saving a Configuration](#)” on page 251 respectively in Chapter 3.*

design object

A design object is a set of files and directories that represents one aspect of a design. In Pyxis Project Manager, these files and directories appear as a single object with a unique icon to represent the object’s type. When you perform operations on a design object, you need not remember which files and directories compose the object. Pyxis Project Manager treats them as a single object. *See also “[Getting Information about a Version](#)” on page 198.*

encapsulation

Encapsulation is the process of integrating design tools and data into the Pyxis Project Manager. Encapsulation controls how tools and data are used within Pyxis Custom Design flow, without affecting the tool or data itself.

extent box

An extent box is a transparent graphical image that the mouse draws. The extent box is displayed as a dynamic rectangle that represents the outer boundaries of an object. When you move an object, a window for example, a rectangle moves with the mouse pointer while the object remains stationary. When you release the mouse button, the object moves to the location where you placed the dynamic rectangle.

External Rendering Interface (ERI)

The External Rendering Interface is a protocol for the generation of renderings of data objects by reference. ERI allows applications to obtain graphical views of designs from Mentor Graphics EDA tools and other third party data formats for inclusion in documents, viewing, printing, and plotting in the V8.0 environment.

fileset

A fileset is the combination of files and directories that compose that object. The files hold either design data or metadata, which is data that describes the design object. The directory may contain other files or design objects. The Pyxis Project Manager treats the entire fileset as a single object. When you use the Pyxis Project Manager to move, copy, or delete a design object, that object's entire fileset is moved, copied, or deleted. *See also “[Design Object Types](#)” on page 519.*

file system objects

File system objects are the files and directories of which a design object is comprised. *See also “[Design Object Types](#)” on page 519.*

freeze file

A freeze file is a frozen version file that is temporarily added to a design object's fileset when you freeze a version. The file is removed when the version is unfrozen.

frozen version

A frozen version is a version of a design object that cannot be deleted by the version depth mechanism. You use frozen versions to save a particular version. *See also “[Getting Information about a Version](#)” on page 198.*

hard pathname

A hard pathname is a pathname that begins with the slash (/) character. A hard pathname may or may not be interoperable. The network and file system environment within which the hard pathname exists, determines whether it is interoperable.

Integrated Design Management

The Integrated Design Management (iDM) toolkit provides a limited set of simple easy-to-use design management commands from within Pyxis Custom Design flow applications. These commands provide the ability to copy, move, delete, and change the references of your design objects, as well as to view the design hierarchy of your data.

interoperable pathname

An interoperable pathname is a pathname that leads you to the same or equivalent object, within a given context, regardless of which workstation you start from on your network. Additionally, if the pathname leads you to a given object at a given time, it leads you to the same object again at any later time.

leaf name

A leaf name is the portion of a pathname after the last “/”. For example, the leaf of pathname `/usr/tmp/ginko/doc_mgc` is `doc_mgc`.

lock

A lock is a mechanism that Pyxis Custom Design flow provides to avoid concurrency conflict among several users trying to access a single design object. Pyxis Custom Design flow applications provide two kinds of locks: read locks and write locks.

When your application read-locks a design object, every user can read, but not edit, that design object. This lock is also known as a shared lock because several applications can read-lock a design object simultaneously. The object remains read-locked until all of the applications release their locks.

When your application write-locks a design object, you can edit that design object. In addition, others can read it, but no others can edit it or lock it. To edit a design object, the application must first write-lock it. This lock is also known as an exclusive lock because only one application can write-lock a design object, and when a design object is write-locked, no other application can lock it.

MGC working directory

The MGC working directory is the pathname of the internal working directory that is appended to the beginning of relative names during pathname resolution.

metadata

Metadata is information about a design object, which is contained in the design object's attribute file. Some of the information found in the attribute file consists of references, properties, and versions. If the design object is locked or frozen, this information is also present.

object browser dialog

The object browser dialog is a dialog box control that allows navigation to and selection of design objects. The application that displays the object browser dialog can filter the navigator's display to show only design objects of particular types.

object property

An object property is a name and value pair that stores user or tool defined information about the design object. Object properties are versioned to reflect differences as the design object evolves, but they propagate forward from the current version to later versions, until they are explicitly changed or deleted in the current version. *See also “[Adding an Object Property](#)” on page 222.*

primary entry

A primary entry is a configuration entry that *you* add to a configuration, as opposed to secondary entries, which the Pyxis Project Manager adds during a build operation. The entry is only a pointer to the design object version. *See also “[Specifying the Build Rules](#)” on page 242 in Chapter 3.*

property

A property is a name and value pair that stores information about a design object, a version, or a reference. An example is the name of the engineer assigned to the project, paired with his phone extension, as in *Joe Smith, ext. 2449*.

protection

A design object’s protection refers to the granting of access permissions to other users.

pruned

Pruning is the automatic deletion of design object versions which exist beyond the current version depth. *See also “[Getting Information about a Version](#)” on page 198.*

Pyxis Project Manager

Pyxis Project Manager is a graphical, easy-to-use interface to your design data. Pyxis Project Manager supports three primary tasks: design navigation, tool invocation, and design data configuration management.

Pyxis Registrar

The Pyxis Registrar is an application used to register (declare) tool and data types, so that the Pyxis Project Manager can recognize them. The Pyxis Registrar is a graphical, easy-to-use application that does not require knowledge of C++ to perform the registration process. For more information about the Pyxis Registrar, refer to the *Pyxis Registrar User’s and Reference Manual* on SupportNet.

qualification script

A qualification script is a required AMPLE script that runs when you invoke a tool within the Pyxis Project Manager, before the tool’s actual executable code runs. The qualification script gathers and evaluates tool arguments, validates tool invocation, and enforces workflow policies or procedures (if desired).

reference

A reference is a pointer from one design object to another. Design tools use references to associate design objects.

reference mode

When the references of a design object are displayed in a navigator window, the navigator window is in reference mode. You are only in reference mode after you have explored the references of a design object by using the Explore References command. When you are in reference mode, the navigator title bar displays the absolute pathname of the design object whose references are displayed in the navigator window, followed by a “@” indicating reference navigation.

reference handle

A reference handle is an integer that uniquely identifies the reference. *See also “Navigation” on page 55.*

reference network

A reference network is the set of design objects that are related through references. The reference network of a design object includes the design object itself and all other design objects that can be reached by traversing references. A reference network is analogous to the design object's containment hierarchy. Using the Pyxis Project Manager navigator, you can traverse the reference network, one path at a time. When building configurations, the Pyxis Project Manager can traverse the entire reference network of a design object in one operation. *See also “Navigation” on page 55.*

reference property

A reference property is a name/value pair that contains information about the reference. The “creating_tool” property identifies the tool that “owns” the reference. Reference properties propagate with the reference from the current version to later versions, until they are explicitly changed or deleted in the current version. *See also “Adding an Object Property” on page 222, and “Navigation” on page 55 respectively.*

reference state

A state is a condition that applies to references and configuration entries. There are three possible states that a reference or configuration entry may possess:

- Current
Current references and configuration entries always point to the current version of the design object that they reference. As the design object evolves, the reference or configuration entry remains set to that current version.
- Fixed
Fixed references and configuration entries always point to the version of the design object to which they were originally set. As the design object evolves, the reference or configuration entry remains set to the same version.
- Read-only (applies to references only)
A read-only reference points to the current version of the target design object, but the object cannot be edited when accessed through this reference, regardless of the access permissions of the referenced object.

See also “Navigation” on page 55.

reference window

The reference window is a bounded area bordered by a rectangular box within which a selected design object's references are displayed and modified. The reference window displays exactly the same items as the navigator does when a design object is selected and its references are explored. *See also “Navigation” on page 55.*

registration

Registration is the process of defining data and tool types. Registration is done using the Pyxis Registrar.

relative pathname

A relative pathname is a pathname that is relative to the MGC working directory. The MGC working directory is the value returned by the Pyxis Project Manager toolkit function `$$get_working_directory()`.

release

A release is a protected copy of a configuration. Protected means that the “released” attribute is assigned to every versioned object in the release directory. As a result, the Pyxis Project Manager does not allow you to create new versions of these released design objects. If you want to open and to evolve these design objects, you must copy the released configuration to another directory. *See also “[Opening a New Configuration](#)” on page 239, and “[Saving a Configuration](#)” on page 251 respectively in Chapter 3.*

re-targetable entry

A re-targetable entry is a configuration entry that can be released or copied to a separate location, other than the location specified for the release or copy operation. An entry is re-targetable if its parent container is not part of the configuration.

revert

Reverting is the process of making the previous version of a design object current, by deleting the current version. *See also “[Getting Information about a Version](#)” on page 198.*

secondary entry

A secondary entry is an entry that the Pyxis Project Manager adds to a configuration during a build. Secondary entries are added based on the build rules of the primary entries. You can set the build rules of primary entries to either include or exclude secondary entries from a configuration. *See also “[Specifying the Build Rules](#)” on page 242 in Chapter 3.*

session window

The session window is the outermost bounded area bordered by a rectangle box. It is within this area that you operate the Pyxis Project Manager.

soft pathname

A soft pathname is a pathname that begins with a soft prefix. Because a soft pathname must begin with a soft prefix, it follows that the initial characters of a soft pathname must be a dollar sign (\$). A soft pathname is not in and of itself interoperable.

soft prefix

A soft prefix is a string that begins with a dollar sign (\$), contains one or more capital letters, digits, or underscore characters. A soft prefix precedes all other characters in a pathname, and is followed by a slash. A soft prefix has an associated value which can be substituted in its place to produce a hard pathname.

status code

A status code is an integer that corresponds to an error message. The status code is returned to the status code stack after a function executes. The status code typically refers to the number at the top of the status code stack. A status code of 0 means no error occurred.

status code stack

A status code stack is a list of status codes.

status message

A status message is a vector of strings that correspond to the status codes on the status code stack.

target path

The target path is the location that a re-targetable configuration entry is released or copied to during a copy or release operation.

termination script

A termination script is an optional AMPLE script that runs in the Pyxis Project Manager session after the design tool has terminated. This script performs cleanup functions required by the design tool.

tool

A tool is software created for a specific task. An example of a Mentor Graphics tool is Pyxis Schematic. The Pyxis Project Manager represents tools using icons. *See also “[Rearranging the Toolbox Search Path](#)” on page 168.*

toolbox

A toolbox is a directory that contains tool viewpoints.

toolbox search path

The toolbox search path is a group of pathnames that specify the order in which the Pyxis Project Manager searches toolboxes when populating the tools window. Given two tool viewpoints with the same name but in different toolboxes, the first tool viewpoint found is the one displayed. *See also “[Rearranging the Toolbox Search Path](#)” on page 168.*

tool invocation

Tool invocation is the process of opening a tool. In the Pyxis Project Manager, you can invoke tools in the navigator, by selecting and opening a design object using the navigator popup menu item **Open**, and in the Tools window by double-clicking a tool icon.

tools window

The tools window is a bounded area bordered by a rectangular box within which are displayed, as icons, all the tools that may be invoked from the Pyxis Project Manager session. *See also “[Rearranging the Toolbox Search Path](#)” on page 168.*

tool viewpoint

A tool viewpoint is a special design object that provides the Pyxis Project Manager interface to a design tool. A tool viewpoint defines the manner in which a design tool invokes and terminates

from the Pyxis Project Manager. You can create multiple tool viewpoints to represent the same tool in the Pyxis Project Manager, with each invoking the tool in a different way.

trash window

The tools window is a bounded area bordered by a rectangular box within which are displayed design objects that have been dragged to the trash can. When the trash is emptied, this window becomes empty, indicating that the objects have been permanently deleted.

type

A description of the attributes of a design object or tool viewpoint, as the attributes exist in the Pyxis Project Manager. Type information includes the filename extensions for the design object's fileset and the icon that the Pyxis Project Manager uses to display the object. *See also “[Design Object Types](#)” on page 519.*

type registry

A type registry is a file that contains type definitions for design objects and tool viewpoints used in the Pyxis Project Manager.

version

A single saved state of a design object. When you change a design object's data and save it to disk, you create a new version of the design object.

version branch

Version branching refers to the version model which allows multiple versions to be derived from a single version. In version models that do not allow version branching, every version, except for the initial version, derives from exactly one version.

version depth

The number of versions that a design object keeps of itself. You can set version depth from one to infinity. *See also “[Getting Information about a Version](#)” on page 198.*

version depth mechanism

The version depth mechanism deletes versions that are beyond the default version depth for a design object. For example, if the default version depth for a design object is 2, the version depth mechanism allows for only two versions to be kept. As the design object evolves, versions beyond 2 are deleted, except for those that are frozen. *See also “[Getting Information about a Version](#)” on page 198.*

versioned design object

A versioned design object is a design object that can have multiple versions. *See also “[Getting Information about a Version](#)” on page 198.*

version property

A version property is a name and value pair that stores information about the design object version. Version properties do not propagate from the current version forward. *See also “[Adding an Object Property](#)” on page 222, and “[Getting Information about a Version](#)” on page 198 respectively.*

version pruning

Version pruning is the deletion of a design object's previous versions based on the specified version depth. Version pruning occurs when a new version is created and old versions exist that exceed the version depth. *See also “[Getting Information about a Version](#)” on page 198.*

Third-Party Information

Open source and third-party software may be included in the Pyxis products.

For third-party information, refer to [*Third-Party Software for Pyxis Products*](#).

End-User License Agreement

The latest version of the End-User License Agreement is available on-line at:
www.mentor.com/eula

IMPORTANT INFORMATION

USE OF ALL SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE PRODUCTS. USE OF SOFTWARE INDICATES CUSTOMER'S COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.

END-USER LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Software (as defined in Section 2) and hardware (collectively "Products") between the company acquiring the Products ("Customer"), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity ("Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, in the case of Software received electronically, certify destruction of Software and all accompanying items within five days after receipt of Software and receive a full refund of any license fee paid.

1. ORDERS, FEES AND PAYMENT.

- 1.1. To the extent Customer (or if agreed by Mentor Graphics, Customer's appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement (each an "Order"), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement, any applicable addenda and the applicable quotation, whether or not those documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order or presented in any electronic portal or automated order management system, whether or not required to be electronically accepted, will not be effective unless agreed in writing and physically signed by an authorized representative of Customer and Mentor Graphics.
- 1.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will state separately in the applicable invoice. Unless timely provided with a valid certificate of exemption or other evidence that items are not taxable, Mentor Graphics will invoice Customer for all applicable taxes including, but not limited to, VAT, GST, sales tax, consumption tax and service tax. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer's sole responsibility. If Customer appoints a third party to place purchase orders and/or make payments on Customer's behalf, Customer shall be liable for payment under Orders placed by such third party in the event of default.
- 1.3. All Products are delivered FCA factory (Incoterms 2010), freight prepaid and invoiced to Customer, except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics retains a security interest in all Products delivered under this Agreement, to secure payment of the purchase price of such Products, and Customer agrees to sign any documents that Mentor Graphics determines to be necessary or convenient for use in filing or perfecting such security interest. Mentor Graphics' delivery of Software by electronic means is subject to Customer's provision of both a primary and an alternate e-mail address.

2. **GRANT OF LICENSE.** The software installed, downloaded, or otherwise acquired by Customer under this Agreement, including any updates, modifications, revisions, copies, documentation and design data ("Software") are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software solely: (a) in machine-readable, object-code form (except as provided in Subsection 5.2); (b) for Customer's internal business purposes; (c) for the term of the license; and (d) on the computer hardware and at the site authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Customer may have Software temporarily used by an employee for telecommuting purposes from locations other than a Customer office, such as the employee's residence, an airport or hotel, provided that such employee's primary place of employment is the site where the Software is authorized for use. Mentor Graphics' standard policies and programs, which vary depending on Software, license fees paid or services purchased, apply to the following: (a) relocation of Software; (b) use of Software, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be technically implemented through the use of authorization codes or similar devices); and (c) support services provided, including eligibility to receive telephone support, updates, modifications, and revisions. For the avoidance of doubt, if Customer provides any feedback or requests any change or enhancement to Products, whether in the course of receiving support or consulting services, evaluating Products, performing beta testing or otherwise, any inventions, product improvements, modifications or developments made by Mentor Graphics (at Mentor Graphics' sole discretion) will be the exclusive property of Mentor Graphics.

3. **ESC SOFTWARE.** If Customer purchases a license to use development or prototyping tools of Mentor Graphics' Embedded Software Channel ("ESC"), Mentor Graphics grants to Customer a nontransferable, nonexclusive license to reproduce and distribute executable

files created using ESC compilers, including the ESC run-time libraries distributed with ESC C and C++ compiler Software that are linked into a composite program as an integral part of Customer's compiled computer program, provided that Customer distributes these files only in conjunction with Customer's compiled computer program. Mentor Graphics does NOT grant Customer any right to duplicate, incorporate or embed copies of Mentor Graphics' real-time operating systems or other embedded software products into Customer's products or applications without first signing or otherwise agreeing to a separate agreement with Mentor Graphics for such purpose.

4. **BETA CODE.**

- 4.1. Portions or all of certain Software may contain code for experimental testing and evaluation (which may be either alpha or beta, collectively "Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. Mentor Graphics may choose, at its sole discretion, not to release Beta Code commercially in any form.
- 4.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer's use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer's evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.
- 4.3. Customer agrees to maintain Beta Code in confidence and shall restrict access to the Beta Code, including the methods and concepts utilized therein, solely to those employees and Customer location(s) authorized by Mentor Graphics to perform beta testing. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer's feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 4.3 shall survive termination of this Agreement.

5. **RESTRICTIONS ON USE.**

- 5.1. Customer may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Customer shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. Customer shall not make Products available in any form to any person other than Customer's employees and on-site contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Products and ensure that any person permitted access does not disclose or use Products except as permitted by this Agreement. Customer shall give Mentor Graphics written notice of any unauthorized disclosure or use of the Products as soon as Customer becomes aware of such unauthorized disclosure or use. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive any source code from Software. Log files, data files, rule files and script files generated by or for the Software (collectively "Files"), including without limitation files containing Standard Verification Rule Format ("SVRF") and Tcl Verification Format ("TVF") which are Mentor Graphics' trade secret and proprietary syntaxes for expressing process rules, constitute or include confidential information of Mentor Graphics. Customer may share Files with third parties, excluding Mentor Graphics competitors, provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Customer may use Files containing SVRF or TVF only with Mentor Graphics products. Under no circumstances shall Customer use Products or Files or allow their use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Products, or disclose to any third party the results of, or information pertaining to, any benchmark.
- 5.2. If any Software or portions thereof are provided in source code form, Customer will use the source code only to correct software errors and enhance or modify the Software for the authorized use. Customer shall not disclose or permit disclosure of source code, in whole or in part, including any of its methods or concepts, to anyone except Customer's employees or on-site contractors, excluding Mentor Graphics competitors, with a need to know. Customer shall not copy or compile source code in any manner except to support this authorized use.
- 5.3. Customer may not assign this Agreement or the rights and duties under it, or relocate, sublicense, or otherwise transfer the Products, whether by operation of law or otherwise ("Attempted Transfer"), without Mentor Graphics' prior written consent and payment of Mentor Graphics' then-current applicable relocation and/or transfer fees. Any Attempted Transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics' option, result in the immediate termination of the Agreement and/or the licenses granted under this Agreement. The terms of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer's permitted successors in interest and assigns.
- 5.4. The provisions of this Section 5 shall survive the termination of this Agreement.

6. **SUPPORT SERVICES.** To the extent Customer purchases support services, Mentor Graphics will provide Customer with updates and technical support for the Products, at the Customer site(s) for which support is purchased, in accordance with Mentor Graphics' then current End-User Support Terms located at <http://supportnet.mentor.com/supportterms>.

7. **LIMITED WARRANTY.**

- 7.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Products, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Products will meet Customer's requirements or that operation of Products will be uninterrupted or error free. The

warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Software under an Order and does not renew or reset, for example, with the delivery of (a) Software updates or (b) authorization codes or alternate Software under a transaction involving Software re-mix. This warranty shall not be valid if Products have been subject to misuse, unauthorized modification, improper installation or Customer is not in compliance with this Agreement. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF THE PRODUCTS TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF THE PRODUCTS THAT DO NOT MEET THIS LIMITED WARRANTY. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) PRODUCTS PROVIDED AT NO CHARGE; OR (C) BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."

- 7.2. THE WARRANTIES SET FORTH IN THIS SECTION 7 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO PRODUCTS PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.
8. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL MENTOR GRAPHICS' OR ITS LICENSORS' LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT RECEIVED FROM CUSTOMER FOR THE HARDWARE, SOFTWARE LICENSE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 8 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.
9. **HAZARDOUS APPLICATIONS.** CUSTOMER ACKNOWLEDGES IT IS SOLELY RESPONSIBLE FOR TESTING ITS PRODUCTS USED IN APPLICATIONS WHERE THE FAILURE OR INACCURACY OF ITS PRODUCTS MIGHT RESULT IN DEATH OR PERSONAL INJURY ("HAZARDOUS APPLICATIONS"). EXCEPT TO THE EXTENT THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF MENTOR GRAPHICS PRODUCTS IN OR FOR HAZARDOUS APPLICATIONS. THE PROVISIONS OF THIS SECTION 9 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.
10. **INDEMNIFICATION.** CUSTOMER AGREES TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE OR LIABILITY, INCLUDING ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH THE USE OF MENTOR GRAPHICS PRODUCTS IN OR FOR HAZARDOUS APPLICATIONS. THE PROVISIONS OF THIS SECTION 10 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.
11. **INFRINGEMENT.**
 - 11.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Product acquired by Customer hereunder infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay costs and damages finally awarded against Customer that are attributable to such action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.
 - 11.2. If a claim is made under Subsection 11.1 Mentor Graphics may, at its option and expense: (a) replace or modify the Product so that it becomes noninfringing; (b) procure for Customer the right to continue using the Product; or (c) require the return of the Product and refund to Customer any purchase price or license fee paid, less a reasonable allowance for use.
 - 11.3. Mentor Graphics has no liability to Customer if the action is based upon: (a) the combination of Software or hardware with any product not furnished by Mentor Graphics; (b) the modification of the Product other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of the Product as part of an infringing process; (e) a product that Customer makes, uses, or sells; (f) any Beta Code or Product provided at no charge; (g) any software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; or (h) infringement by Customer that is deemed willful. In the case of (h), Customer shall reimburse Mentor Graphics for its reasonable attorney fees and other costs related to the action.
 - 11.4. THIS SECTION 11 IS SUBJECT TO SECTION 8 ABOVE AND STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS, AND CUSTOMER'S SOLE AND EXCLUSIVE REMEDY, FOR DEFENSE, SETTLEMENT AND DAMAGES, WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY PRODUCT PROVIDED UNDER THIS AGREEMENT.
12. **TERMINATION AND EFFECT OF TERMINATION.**
 - 12.1. If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized term. Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement immediately upon written notice if Customer: (a) exceeds the scope of the license or otherwise fails to comply with the licensing or confidentiality provisions of this Agreement, or (b) becomes insolvent, files a bankruptcy petition, institutes proceedings for liquidation or winding up or enters into an agreement to assign its assets for the benefit of creditors. For any other material breach of any

provision of this Agreement, Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement upon 30 days written notice if Customer fails to cure the breach within the 30 day notice period. Termination of this Agreement or any license granted hereunder will not affect Customer's obligation to pay for Products shipped or licenses granted prior to the termination, which amounts shall be payable immediately upon the date of termination.

- 12.2. Upon termination of this Agreement, the rights and obligations of the parties shall cease except as expressly set forth in this Agreement. Upon termination, Customer shall ensure that all use of the affected Products ceases, and shall return hardware and either return to Mentor Graphics or destroy Software in Customer's possession, including all copies and documentation, and certify in writing to Mentor Graphics within ten business days of the termination date that Customer no longer possesses any of the affected Products or copies of Software in any form.
13. **EXPORT.** The Products provided hereunder are subject to regulation by local laws and United States ("U.S.") government agencies, which prohibit export, re-export or diversion of certain products, information about the products, and direct or indirect products thereof, to certain countries and certain persons. Customer agrees that it will not export or re-export Products in any manner without first obtaining all necessary approval from appropriate local and U.S. government agencies. If Customer wishes to disclose any information to Mentor Graphics that is subject to any U.S. or other applicable export restrictions, including without limitation the U.S. International Traffic in Arms Regulations (ITAR) or special controls under the Export Administration Regulations (EAR), Customer will notify Mentor Graphics personnel, in advance of each instance of disclosure, that such information is subject to such export restrictions.
14. **U.S. GOVERNMENT LICENSE RIGHTS.** Software was developed entirely at private expense. The parties agree that all Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to U.S. FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. government or a U.S. government subcontractor is subject solely to the terms and conditions set forth in this Agreement, which shall supersede any conflicting terms or conditions in any government order document, except for provisions which are contrary to applicable mandatory federal laws.
15. **THIRD PARTY BENEFICIARY.** Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, Microsoft Corporation and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.
16. **REVIEW OF LICENSE USAGE.** Customer will monitor the access to and use of Software. With prior written notice and during Customer's normal business hours, Mentor Graphics may engage an internationally recognized accounting firm to review Customer's software monitoring system and records deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FlexNet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. The provisions of this Section 16 shall survive the termination of this Agreement.
17. **CONTROLLING LAW, JURISDICTION AND DISPUTE RESOLUTION.** The owners of certain Mentor Graphics intellectual property licensed under this Agreement are located in Ireland and the U.S. To promote consistency around the world, disputes shall be resolved as follows: excluding conflict of laws rules, this Agreement shall be governed by and construed under the laws of the State of Oregon, U.S., if Customer is located in North or South America, and the laws of Ireland if Customer is located outside of North or South America. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of the courts of Portland, Oregon when the laws of Oregon apply, or Dublin, Ireland when the laws of Ireland apply. Notwithstanding the foregoing, all disputes in Asia arising out of or in relation to this Agreement shall be resolved by arbitration in Singapore before a single arbitrator to be appointed by the chairman of the Singapore International Arbitration Centre ("SIAC") to be conducted in the English language, in accordance with the Arbitration Rules of the SIAC in effect at the time of the dispute, which rules are deemed to be incorporated by reference in this section. Nothing in this section shall restrict Mentor Graphics' right to bring an action (including for example a motion for injunctive relief) against Customer in the jurisdiction where Customer's place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.
18. **SEVERABILITY.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.
19. **MISCELLANEOUS.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements. Some Software may contain code distributed under a third party license agreement that may provide additional rights to Customer. Please see the applicable Software documentation for details. This Agreement may only be modified in writing, signed by an authorized representative of each party. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.