

Milkyway™ Environment Extension Language Reference Manual

Version W-2004.12, December 2004

Comments?

E-mail your comments about Synopsys
documentation to doc@synopsys.com

SYNOPSYS®

Copyright Notice and Proprietary Information

Copyright © 2003 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

"This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of _____ and its employees. This is copy number _____. "

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Registered Trademarks (®)

Synopsys, AMPS, Arcadia, C Level Design, C2HDL, C2V, C2VHDL, Cadabra, Calaveras Algorithm, CATS, CoCentric, COSSAP, CSim, DelayMill, Design Compiler, DesignPower, DesignWare, Device Model Builder, EPIC, Formality, HSPICE, Hypermodel, I, iN-Phase, InSpecs, in-Sync, LEDA, MAST, Meta, Meta-Software, ModelAccess, ModelExpress, ModelTools, PathBlazer, PathMill, Photolynx, Physical Compiler, PowerArc, PowerMill, PrimeTime, RailMill, Raphael, RapidScript, Saber, SIVL, SmartLogic, SNUG, SolvNet, Stream Driven Simulator, Superlog, System Compiler, Testify, TetraMAX, TimeMill, TMA, Vera, and Virtual Stepper are registered trademarks of Synopsys, Inc.

Trademarks (™)

abraCAD, abraMAP, Active Parasitics, AFGen, Apollo, Apollo II, Apollo-DPII, Apollo-GA, ApolloGAI, Astro, Astro-Rail, Astro-Xtalk, Aurora, AvanTestchip, AvanWaves, BCView, Behavioral Compiler, BOA, BRT, Cedar, ChipPlanner, Circuit Analysis, Columbia, Columbia-CE, Comet 3D, Cosmos, CosmosEnterprise, CosmosLE, CosmosScope, CosmosSE, Cyclelink, Davinci, DC Expert, DC Expert Plus, DC Professional, DC Ultra, DC Ultra Plus, Design Advisor, Design Analyzer, DesignerHDL, DesignTime, DFM-Workbench, DFT Compiler, Direct RTL, Direct Silicon Access, DW8051, DWPCI, Dynamic-Macromodeling, Dynamic Model Switcher, ECL Compiler, ECO Compiler, EDAnavigator, Encore, Encore PQ, Evaccess, ExpressModel, Floorplan Manager, Formal Model Checker, FormalVera, FoundryModel, FPGA Compiler II, FPGA Express, Frame Compiler, Frameway, Galaxy, Gatran, HDL Advisor, HDL Compiler, Hercules, Hercules-Explorer, Hercules-II, Hierarchical Optimization Technology, High Performance Option, HotPlace, HSPICE-Link, iN-Tandem, Integrator, Interactive Waveform Viewer, iQBus, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, JVXtreme, Liberty, Libra-Passport, Library Compiler, Libra-Visa, LRC, Mars, Mars-Rail, Mars-Xtalk, Medici, Metacapture, Metacircuit, Metamanager, Metamixsim, Milkyway, ModelSource, Module Compiler, MS-3200, MS-3400, NanoSim, Nova Product Family, Nova-ExploreRTL, Nova-Trans, Nova-VeriLint, Nova-VHDLint, OpenVera, Optimum Silicon, Orion_ec, Parasitic View, Passport, Planet, Planet-PL, Planet-RTL, Polaris, Polaris-CBS, Polaris-MT, Power Compiler, PowerCODE, PowerGate, ProFPGA, Progen, Prospector, Proteus OPC, Protocol Compiler, PSMGen, Raphael-NES, RoadRunner, RTL Analyzer, Saturn, ScanBand, Schematic Compiler, Scirocco, Scirocco-i, Shadow Debugger, Silicon Blueprint, Silicon Early Access, SinglePass-SoC, Smart Extraction, SmartLicense, SmartModel Library, Softwire, Source-Level Design, Star, Star-DC, Star-MS, Star-MTB, Star-Power, Star-Rail, Star-RC, Star-RCXT, Star-Sim, Star-Sim XT, Star-Time, Star-XP, SWIFT, Taurus, Taurus-Device, Taurus-Layout, Taurus-Lithography, Taurus-OPC, Taurus-Process, Taurus-Topography, Taurus-Visual, Taurus-Workbench, The Power in Semiconductors, TimeSlice, TimeTracker, Timing Annotator, TopoPlace, TopoRoute, Trace-On-Demand, True-Hspice, TSUPREM-4, TymeWare, VCS, VCS Express, VCSI, Venus, Verification Portal, VFormal, VHDL Compiler, VHDL System Simulator, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (SM)

DesignSphere, MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.
AMBA is a trademark of ARM Limited. ARM is a registered trademark of ARM Limited.
All other product or company names may be trademarks of their respective owners.

Printed in the U.S.A.

Document Order Number: 00000-000 TA
Milkyway™ Environment Extension Language Reference Manual, version V-2004.06

Contents

What's New in This Release	xxiv
About This Guide.....	xxv
Customer Support.....	xxix
1. Using Tcl in the Milkyway Environment	
Scheme and Tcl in Milkyway	1-3
Tcl Support for Tools Based on Milkyway	1-5
Tcl-Equivalent Command Guidelines	1-7
Scheme and Tcl Syntax and Semantics	1-8
Tcl-equivalent Command Limitations.....	1-10
Introduction to Synopsys Tcl	1-11
Synopsys Tcl Collections in Milkyway	1-12
Attribute Support	1-14
Synopsys Tcl Help, Man Pages, and User Message Handling	1-16
Synopsys Tcl Help.....	1-16
Man Page Support.....	1-17

Enhanced Milkyway Message Handling	1-18
Scheme and Tcl Execution Modes	1-19
Milkyway Library Search Management	1-22
Setting search_path and link_library	1-23
Logic Library Search Order Variables	1-24
Ordered .db File List	1-26
Reference Library Management	1-27
Synopsys Tcl Commands	1-28
Scheme Commands Not Supported in Tcl	1-70
2. Scheme Fundamentals	
Lexical Conventions.	2-3
Identifiers	2-3
Numbers	2-4
Characters	2-5
White Space and Comments	2-5
Other Notations	2-6
Basic Concepts	2-7
Variables	2-7
The Evaluator.	2-8
Errors	2-10
Interrupts	2-11
True and False	2-13
Expressions.	2-13
Primitive Expression Types	2-13

Derived Expression Types	2-16
Definitions	2-19
Top-Level Definitions	2-19
Internal Definitions	2-20
Boolean Procedures	2-21
not	2-21
boolean?	2-22
and	2-22
or	2-23
Equivalence Predicates	2-23
eq?	2-23
eqv?	2-24
equal?	2-25
Pairs and Lists	2-26
pair?	2-26
list?	2-27
null?	2-28
cons	2-28
car	2-29
cdr	2-30
set-car!	2-30

set-cdr!	2-31
length.	2-32
append	2-32
Membership functions	2-33
Association functions	2-34
Symbol Predicates	2-35
symbol?	2-35
Numbers	2-36
Number Equality Predicates	2-37
Arithmetic Functions	2-38
Number Conversion	2-39
char?	2-40
char->integer	2-40
integer->char	2-41
char-upcase	2-41
char-downcase	2-42
String Functions	2-42
string?	2-42
string-length	2-43
string-ref	2-43
string-set!	2-44
substring	2-44

string-append	2-45
string->list	2-45
list->string	2-46
string-copy	2-46
string-fill!	2-47
string->float	2-47
string->integer	2-48
number->string	2-48
Control Features	2-49
procedure?	2-49
apply	2-49
map	2-50
for-each	2-51
call-with-current-continuation	2-52
Input and Output	2-54
input-port?	2-54
output-port?	2-54
open-input-file	2-55
open-output-file	2-55
close-input-port	2-56
close-output-port	2-57

read-char	2-57
eof-object?	2-58
read-string	2-58
display	2-59
newline	2-60
write-char	2-60
System Interface	2-61
load	2-61
3. General Extensions	
bit-and	3-2
bit-or	3-2
bit-not	3-2
CONSTANT	3-3
FileExists?	3-4
FileIsReadable?	3-5
FileIsWritable?	3-6
FileIsExecutable?	3-6
getDisplayMode	3-7
getenv	3-8
getInstallPath	3-9
getIntegVersion	3-9

getInterruptMask	3-10
getInterruptPollInterval	3-10
getProgramName	3-11
getSubVersion	3-11
getVersion	3-12
getVersionIdent	3-13
getwd	3-13
interrupt?	3-14
loadModule	3-14
loadModuleBindImmediate	3-16
queryModules	3-16
rand	3-17
releaseModule	3-18
dbSetBckFile	3-18
setDisplayMode	3-19
setInterruptMask	3-20
setInterruptPollInterval	3-21
dbSetPriv	3-21
signal-interrupt-error	3-22
sleep	3-24
srand	3-24

system	3-25
time	3-25
4. Scheme Database Extensions	
Understanding the Synopsys Database	4-2
References Between Database Objects	4-3
Naming Guidelines	4-3
Case Sensitivity	4-6
Working with Cell Libraries	4-7
Cell Libraries	4-8
Naming Libraries	4-9
Accessing Libraries	4-10
Specifying Libraries	4-10
Scheme Extensions for Working With Libraries	4-12
dbAddLibSchemeFile	4-13
dbAddRefLib	4-14
dbAddIOText	4-15
dbAllowToAddAllPGIOText	4-16
dbClearCellDifferentialGroup.	4-17
dbCloseLib	4-17
dbConvertPortToDiodePort	4-19
dbCopyLib	4-20
dbCreateLib.	4-21

dbCreatePcellDefinition	4-22
dbDefineDifferentialGroup	4-23
dbDeleteLibAttachedFile	4-24
dbDeletePcellDefinition	4-25
dbDeleteRefLibByName	4-26
dbDowngradeLib	4-28
dbDumpCellDifferentialGroup	4-28
dbEliminatePins	4-29
dbGetAllLibCellInfo	4-30
dbGetMaxMetalRoutingLayer	4-31
dbGetOneLibCellInfo	4-32
dbListUnitRange	4-32
dbOpenLib	4-33
dbOpenLibrary	4-33
dbRenameLib	4-35
dbReplaceRefLibByName	4-36
dbReplaceTechFile	4-38
dbSetHierRefLibMode	4-39
dbSetIOPad	4-40
dbSetMacroCellOrient	4-41
dbSetStdCell	4-43

dbSetTechFileName	4-43
dbStoreLibAttachedFile	4-44
Working with Cells	4-45
Cell Access	4-48
Specifying Cells	4-49
Scheme Extensions for Working with Cells	4-52
dbAbortCell	4-52
dbChangeAccessMode	4-53
dbCloseCell	4-54
dbCopyCell	4-55
dbCopyCellFromAnotherLib	4-57
dbCopyCellToAnotherLibDisk	4-58
dbDeleteCell	4-60
dbDeleteCellAttachedFile	4-60
dbDeleteInstAndStitch	4-62
dbDumpCell	4-63
dbDumpFreezeRoutingNets	4-63
dbFreezeRoutingOnNets	4-65
dbOpenCell	4-67
dbRenameCell	4-68
dbRollBackCell	4-69

dbSaveCell	4-70
dbStoreCellAttachedFile	4-71
Extensions for Querying the Database	4-73
->?	4-73
->type	4-74
dbCloseQueryFile	4-75
clfDump	4-76
dbFetchContactGeom	4-77
dbFetchLayerIdByName	4-78
dbFetchLayerIdByNumber	4-79
dbFetchLayerIdList	4-80
dbFetchLayerInfo	4-80
dbFetchLayerTypes	4-82
dbFetchObject	4-82
dbFetchObjectField	4-84
dbFetchPinInstByPortInst	4-87
dbFetchScanTypes	4-88
dbFetchTechInfo	4-90
dbFetchTechTypes	4-91
dbGetAllInstFromFence	4-91
dbGetAllInstFromGroup	4-92

dbGetCellInstByName	4-93
clfGetCLFLibName	4-95
clfGetCLFLibId	4-95
dbGetConstraintsCellId	4-95
dbGetCurrentLibId	4-96
dbGetLayerNameFromNumber	4-96
dbGetLayerNumberFromName	4-97
dbGetLibIdByCellId	4-99
dbGetNetByName	4-99
dbGetPinByName	4-101
dbGetPortByName	4-102
dbGetRegionIdByCellInstId	4-103
dbGetRegionPortByName	4-104
dbGetRegionPortDirection	4-105
dbId?	4-105
dbOpenQueryFile	4-106
dbQuery	4-107
Extensions for Modifying the Database	4-108
dbAddCellInstToCellRegion	4-108
dbAddObjectToGroup	4-109
dbClearCellIEEQClass	4-111

dbClearCellEQClass	4-111
dbClearCellEQClass	4-112
dbConnect	4-113
dbConnectPortInstByName	4-115
dbConvertRegionToMacroCell.	4-116
dbCreateCellBoundary	4-117
dbCreateCellBoundaryForCell	4-118
dbCreateCellEQClass.	4-119
dbCreateCellEQClass	4-121
dbCreateCellProperty	4-122
dbCreateCellRegion	4-123
dbCreateCellRow	4-125
dbCreateChildNet	4-126
dbCreateChipPort	4-127
dbCreateContact	4-129
dbCreateError	4-131
dbCreateErrorClass.	4-132
dbCreateGroup	4-134
dbCreateHierNet	4-135
dbCreateHorizontalWire	4-137
dbCreateLeqPortGroup	4-138

dbCreateVerticalWire	4-139
dbCreateNet	4-140
dbCreateNode	4-142
dbCreateParamSet	4-144
dbCreateParentNet	4-144
dbCreatePin	4-146
dbCreatePolygon	4-147
dbCreatePort	4-148
dbCreatePreRoute	4-149
dbCreateProperty	4-151
dbCreatePseudoPath	4-152
dbCreateRectangle	4-153
dbCreateRegionPin	4-155
dbCreateRegionPort	4-156
dbCreateText	4-158
dbCreateWireMaster	4-160
dbDeleteCellInstFromCellRegion	4-161
dbDeleteCellRowDoubleBack	4-162
dbDeleteLcqPortGroup	4-163
dbDeleteObject	4-164
dbDeleteOneLayerRectangle	4-165

dbDeletePin	4-166
dbDeleteProperty	4-167
dbDeletePropertyById	4-167
dbDeletePropertyName	4-168
dbDisconnect	4-169
dbDisconnectPortInstByName	4-171
dbDisplayCellRailProperty	4-172
dbDumpCellPlacement	4-173
dbDumpGPortTable	4-174
dbDumpSingleViewPageNumber	4-175
dbDumpTechFile	4-176
dbDumpViewRecordNumber	4-177
dbDumpViewPageNumber	4-178
dbForceRecompCellBBox	4-178
dbForceRecursiveUpdateAllHierarchy	4-179
dbForceUpdateHierarchy	4-180
db-foreach	4-181
dbMakeWireMasterId	4-190
dbMarkCellType	4-192
dbMilkywayTranslateString	4-193
dbMoveAllNetMember	4-194

dbMoveCellOrigin	4-197
dbPurgeAllEmptyNet	4-198
dbPurgeAllSingleNet	4-199
dbPurgeCellInstByName	4-200
dbPurgeCellInstMaster	4-200
dbPurgeNet	4-201
dbPurgeNetByName	4-202
dbPurgeObject	4-203
dbPurgePin	4-204
dbPurgePort	4-205
dbPurgeRegionPin	4-205
dbPurgeRegionPort	4-206
dbRebuildLib	4-207
dbRemoveObjectFromGroup	4-208
dbReplaceCellInst	4-209
dbReplaceRegionWithInst	4-210
dbResetCellMustJoinPin	4-211
dbSetCellInstECOStatus	4-213
dbSetCellInstECOStatusByName	4-214
dbSetCellInstFixStatusByName	4-216
dbSetCellInstOrigin	4-217

dbSetCellInstStatus	4-218
dbSetCellOrigin	4-219
dbSetCellPortTypeExtraView	4-221
dbSetCellPortTypes	4-222
dbSetCellRegionBBox	4-226
dbSetCellRegionTargetPercent	4-227
dbSetCellRowMirror	4-228
dbSetCellType	4-229
dbSetErrorClassColor	4-230
dbSetErrorClassPattern	4-231
dbSetErrorClassStatus	4-232
dbSetErrorStatus	4-233
dbSetLayoutSubType	4-234
dbSetMaxMetalRoutingLayer	4-236
dbSetNetType	4-237
dbSetNetTypeByName	4-238
dbSetPinAccessDir	4-239
dbSetPinAccessDirByName	4-241
dbSetPinLayerNumber	4-242
dbSetPinStatusFlag1	4-243
dbSetPolygonPointArray	4-244

dbSetPortDirection	4-245
dbSetPortInstNetId	4-246
dbSetSkipRefLibReadLock	4-248
dbSetStdCellOrient	4-248
dbStartCellRowDoubleBack	4-250
dbUpdateCellProperty	4-251
dbUpdateCellRailProperty	4-252
dbRecursiveUpdateAllHierarchy	4-254
dbUpdateHierarchy	4-255
dbUpdateNetStatus	4-255
dbUpdateProperty	4-256
dbVerifyCellEEQClass	4-258
dbVerifyCellLEQClass	4-259
 Extensions for Modifying the Database - Timing	4-259
dbCreateSGEdge	4-259
dbCreateSGPrimaryNode	4-260
dbSetCellPowerValue	4-262
dbSetFringeFactor	4-263
dbSetLayerTimeInfo	4-265
dbSetLModelSubType	4-266

dbSetPathConstraint	4-268
AstroGA Database Extensions	4-271
dbSetCellPattern	4-271
dbAddCellInstToPlanGroupByName	4-273
dbAddNetSubTypeByName	4-275
dbAddPartialScanToScanChain	4-276
dbArchiveCellToLib	4-277
dbArchiveHierDesign	4-282
dbClearMustJoinPortByCell	4-283
dbCreateBaseArray	4-284
dbCreateBaseArrayRow	4-285
axCreateContactArray	4-287
dbCreateErrorComponent	4-290
dbCreateErrorObj	4-292
dbCreateErrorHandler	4-293
dbCreatePath	4-295
axCreateRowRecord	4-297
axCreateTilePatternRecord	4-299
axCreateTrackRecord	4-300
axCreateWireDirRecord	4-302
geOutputPoint	4-302

geOutputLine	4-303
geOutputBBox	4-303
dbPurgeAllPlanGroup	4-303
dbRemoveInstGroupFromScanChain	4-304
dbReplaceCellTextChar	4-306
dbReplaceCellTextStr	4-307
dbReplaceChildCell	4-308
dbRestoreHierDesign	4-309
dbSetCellInstPlacement	4-311
dbSetCellMustJoin2Pin	4-313
dbSetCellMustJoinPinPattern	4-314
dbSetErrorTypeStatus	4-315
dbSetErrorObjStatus	4-316
dbSetNetSubTypeByName	4-317
dbSetScanChainDefaultPriority	4-319

5. Scheme Graphic Editor Extensions

geDeselectAllObject	5-2
geDeselectObject	5-2
geGetEditCell	5-3
geGetEnvValue	5-4
geGetSelectedSet	5-6

hiliteNet	5-6
gelsLayerSelectable	5-7
gelsLayerVisible	5-8
gelsObjectSelectable	5-8
gelsObjectVisible	5-9
geSelectObject	5-10
geSetEnvValue	5-11
geSetLayerSelectable	5-15
geSetLayerVisible	5-16
geSetObjectSelectable	5-17
geSetObjectVisible	5-19
6. Scheme Floorplanner Extensions	
fpGetFPOption	6-2
fpSetFPOption	6-4
7. Synopsys Scheme Examples	
Beginning Scheme Examples	7-2
Example 1 - Find Open Library Name	7-2
Example 2 - Cell Nets	7-2
Example 3 - Objects of Specific Type	7-3
Example 4 - Instances of Open Cell	7-4

Example 5 - Cell Names for a View.....	7-5
Example 6 - Cell Instances Assigned to a Row.....	7-6
Intermediate Scheme Examples	7-7
Example 5 - Bounding Box of Cell Instances	7-7
Example 6 - Fanout Search.....	7-9
Example 7 - Pins and Ports Checker	7-10
 8. Using the Milkyway Scheme Debugger	
Introduction	8-2
Getting Started	8-5
Starting the Debugger	8-5
Interacting With the Debugger	8-5
Setting and Clearing Breakpoints	8-6
Debugging a Program	8-6
Quitting the Debugger	8-7
Debug Commands.....	8-8
File Loading Commands	8-8
file filename	8-8
load [filename]	8-9
Execution Commands	8-9
run [filename]	8-10
step [n]	8-10
next.....	8-11
cont.....	8-11
Breakpoint Commands	8-11

break [filename] line	8-12
clear [filename] line	8-12
clearall	8-12
status	8-13
Other Commands	8-13
assign var exp	8-14
print exp	8-14
search string.	8-14
bsearch string.	8-15
cd dir.	8-15
pwd.	8-16
quit	8-16
exit	8-16

9. Customizing Milkyway

Creating Milkyway Menus and Dialog Boxes.	9-2
Creating Menus by Using Tcl/Tk	9-2
Tcl/Tk Scheme Interface	9-6
Passing Variables to Scheme From Tcl/Tk.	9-7
Creating Menus in Milkyway	9-8
Creating Menus With Scheme	9-8
About the Default Milkyway Tools Menu.	9-10
Adding a New Menu to the Milkyway GUI	9-13
Attached-File Management	9-19
Tcl Attached File Command Overview.	9-21
Specifying Attached-File Link Mode.	9-23
Removing Attached Files by Linking	9-24

Appendix A. Milkyway Database Records

Common Fields	A-2
accessDirection	A-2
allowablePattern	A-3
bbox	A-4
cellFlag	A-5
cellRegionNumber	A-7
direction	A-7
flag	A-8
layer	A-9
localStatus	A-9
pathType	A-10
Pin/Region Pin statusFlag	A-10
routeType	A-11
tile	A-12
transformCode	A-13
Cell Hierarchy Records	A-14
CellInst Record (User View)	A-14
CellInstArray Record	A-17
CellInstMaster Record	A-19
ExpCellInst Record	A-22
ExpPinInst Record	A-24
ExpPortInst Record	A-26
PinInstMaster Record	A-28
PortInst Record	A-31
PortInstMaster Record	A-32

Connectivity Records	A-33
Net Record	A-33
Pin Record	A-36
Port Record	A-40
Database File Records	A-41
CellInfo Record	A-41
LibInfo Record	A-45
Error Cell Records	A-45
ErrorObj Record	A-45
ErrorType Record	A-46
ErrorComponentRecord	A-47
Interregion Global Routing Records	A-49
GLink Record	A-49
Guide Record	A-50
IGRPath Record	A-52
PreRouteSegment Record	A-53
SlicingTreeNode Record	A-56
Placement and Route Records	A-58
Boundary Record	A-58
CellRegion Record	A-59
CellRow Record	A-61
GateSite Record	A-64
GateSiteArray Record	A-64
RegionBlockage Record	A-66
RegionPin Record	A-67

RegionPort Record	A-69
TilePattern Record.....	A-70
WireDirection Record.....	A-71
WireTrack Record	A-71
 Polygon Layout Records	A-73
BPVContactArray Record	A-73
BPVPath Record	A-75
Contact Record	A-76
ContactArray Record	A-77
ExpandedPolygon Record	A-79
HorizontalWire Record.....	A-80
Path Record	A-81
Polygon Record	A-82
Rectangle Record	A-83
RectArray Record	A-84
Text Record	A-85
VerticalWire Record.....	A-86
ViaRegion Record	A-88
WireMaster Record	A-89
 Timing Related Records	A-90
3ValueSet Record	A-90
4ValueSet.....	A-91
6ValueSet.....	A-91
ClkOptData Record	A-92
DelayTable	A-93
PortFunc Record	A-94

PortToPortDelay Record	A-95
PortToPortTable Record	
A-107	
SlackGraphEdge Record	
A-109	
SlackGraphPrimaryNode Record	
A-110	
Other Records	
A-112	
BaseArray Record	
A-112	
CellRow Record	
A-113	
CellEquivalentClass Record	
A-115	
InstPower Record	
A-115	
NetPower Record	
A-116	
PinAccEdge Record	
A-116	
Top Design Format Records	
A-117	
TdfClk Record	
A-117	
TdfFlipChipPad Record	
A-118	
TdfFlipChipMatrix Record	
A-120	

TdfInputOutput Record	A-120
TdfNet Record	A-122
TdfPad Record	A-123
TdfResCap Record	A-125
TdfTimeRequire Record	A-126
Cell Library Format Records	A-127
ClfPadDef Record	A-127
ClfRecord Record	A-128
DataPath Records	A-128
DpsCell Record	A-128
DpsDpcm Record	A-129
DpsSlot Record	A-130
Plan Group Records	A-130
PlanExplode Record	A-130

PlanGroup Record
A-131

PlanGroupPin Record
A-132

Preface

This preface includes the following sections:

- [What's New in This Release](#)
- [About This Guide](#)
- [Customer Support](#)

What's New in This Release

Information about new features, enhancements, changes, known problems, and limitations is available in the *Milkyway Release Notes* in SolvNet.

To see the *Milkyway Release Notes*,

1. Go to the Synopsys Web page at <http://www.synopsys.com> and click SolvNet.
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.)
3. Click Release Notes in the Main Navigation section (on the left), move your pointer to Milkyway, then choose W-2004.12 in the menu that appears.

The W-2004.12 Release Notes appear.

About This Guide

The *Milkyway Environment Extension Language Reference Manual* describes how to use Tcl and Scheme languages with the Milkyway Database.

Synopsys Tcl is the standard language. It is a common interface for tools based on Milkyway and simplifies using standard Tcl for IC design and verification. Milkyway Tcl capability is complemented by a Common Command Interpreter (CCI), an extension layer with enhanced features on top of Tcl. Synopsys Tcl refers to the Tcl language with the Synopsys CCI layer. All Synopsys Tcl commands share the same common code interpreter.

Synopsys Tcl has fewer commands than Scheme. Scheme provides an extensive set of data types and flexible control structures, making it versatile. Because machine dependencies are hidden from the developer, Scheme programs are very portable across implementations of the same Scheme system on two different platforms.

Scheme is a high-level programming language that supports operations on the following objects:

- Strings
- Lists or vectors of objects
- Vectors
- Numbers
- Characters

Audience

Users of this manual must be familiar with physical IC implementation concepts (specifically, standard-cell-based design) and with programming in an interactive extension language.

Related Publications

For additional information about the *Milkyway Environment Extension Language Reference Manual*, see

- Synopsys Online Documentation (SOLD), which is included with the software for CD users or is available to download through the Synopsys Electronic Software Transfer (EST) system
- Documentation on the Web, which is available through SolvNet at <http://solvnet.synopsys.com>
- The Synopsys MediaDocs Shop, from which you can order printed copies of Synopsys documents, at <http://mediadocs.synopsys.com>

You might also want to refer to the following related Synopsys documentation:

- *Milkyway Environment C-API Reference Manual*
- *Milkyway Environment Data Preparation User Guide*
- *Physical Implementation Getting Started*
- Physical Implementation Online Help

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
Courier	Indicates command syntax.
<i>Courier italic</i>	Indicates a user-defined value in Synopsys syntax, such as <i>object_name</i> . (A user-defined value that is not Synopsys syntax, such as a user-defined value in a Verilog or VHDL statement, is indicated by regular text font italic.)
Courier bold	Indicates user input—text you type verbatim—in Synopsys syntax and examples. (User input that is not Synopsys syntax, such as a user name or password you enter in a GUI, is indicated by regular text font bold.)
[]	Denotes optional parameters, such as <i>pin1 [pin2 ... pinN]</i>
	Indicates a choice among alternatives, such as low medium high (This example indicates that you can enter one of three possible values for an option: low, medium, or high.)
-	Connects terms that are read as a single term by the system, such as set_annotated_delay
Control-c	Indicates a keyboard combination, such as holding down the Control key and pressing c.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

SolvNet includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services including software downloads, documentation on the Web, and “Enter a Call to the Support Center.”

To access SolvNet,

1. Go to the SolvNet Web page at <http://solvnet.synopsys.com>.
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.)

If you need help using SolvNet, click SolvNet Help in the Support Resources section.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a call to your local support center from the Web by going to <http://solvnet.synopsys.com> (Synopsys user name and password required), then clicking “Enter a Call to the Support Center.”
- Send an e-mail message to your local support center.
 - E-mail support_center@synopsys.com from within North America.
 - Find other local support center e-mail addresses at http://www.synopsys.com/support/support_ctr.
- Telephone your local support center.
 - Call (800) 245-8005 from within the continental United States.
 - Call (650) 584-4200 from Canada.
 - Find other local support center telephone numbers at http://www.synopsys.com/support/support_ctr.

1

Using Tcl in the Milkyway Environment

This chapter describes the syntax and semantics of the Scheme and Tool Control Language (Tcl) languages and provides help with converting scripts and flows from Scheme to Tcl-equivalent commands. It also introduces Synopsys Tcl commands and offers man page support.

This chapter includes the following sections:

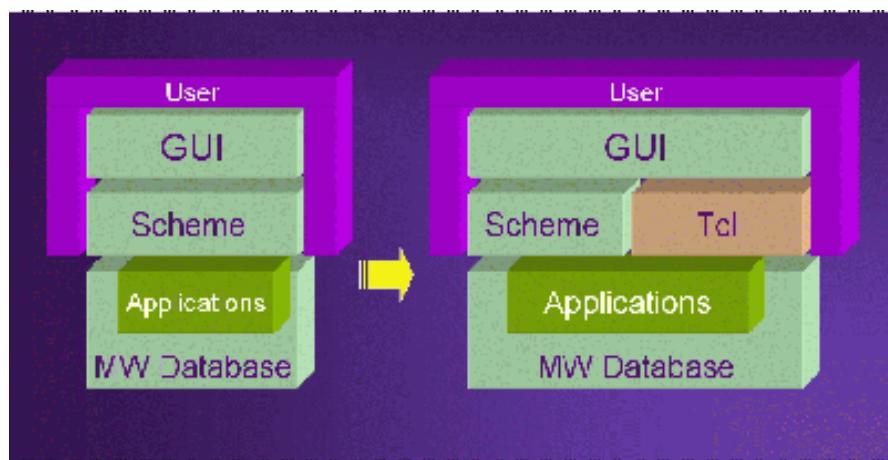
- [Scheme and Tcl in Milkyway](#)
- [Tcl Support for Tools Based on Milkyway](#)
- [Tcl-Equivalent Command Guidelines](#)
- [Introduction to Synopsys Tcl](#)
- [Synopsys Tcl Help, Man Pages, and User Message Handling](#)
- [Scheme and Tcl Execution Modes](#)

- Synopsys Tcl Commands
- Scheme Commands Not Supported in Tcl

Scheme and Tcl in Milkyway

Synopsys supports two extension languages in Milkyway, Scheme and Tcl. Prior to the V-2004.06 release, Milkyway supported only Scheme. The migration to a dual extension language architecture is shown in [Figure 1-1](#).

Figure 1-1 Scheme and Tcl in Milkyway



Tcl and Scheme are high-level scripting languages that enable you to perform tasks more conveniently than with C/C++ (using the stand-alone or dynamic C-API). Tcl and Scheme are *interpreted* languages, allowing you to execute code directly, without compiling and linking (as you do with C/C++). They are also extensible—they allow additional commands to *extend* the Tcl and Scheme languages. Specifically for Milkyway, Synopsys has extended the Tcl and Scheme languages to address the needs of IC design and verification.

Note:

Using the dynamic C-API, you can register a function and make it available in the Milkyway environment as a Scheme command. However, you cannot currently add commands to Tcl through the dynamic C-API.

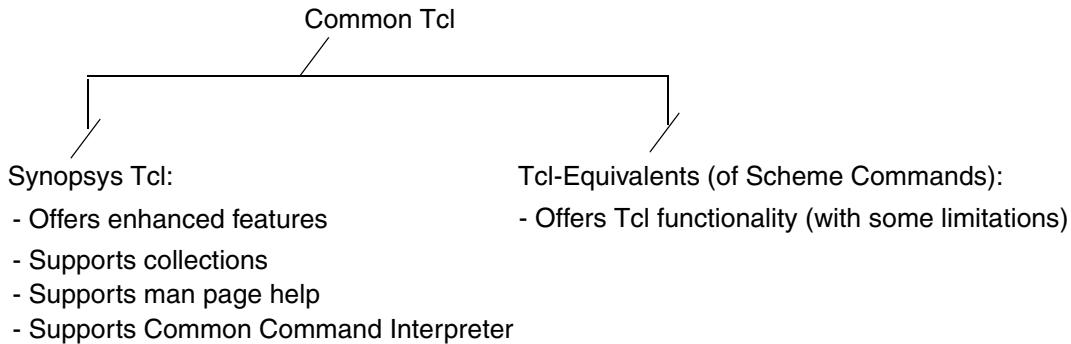
With the introduction of the Milkyway Tcl extension language, most existing Scheme commands have been re-registered as Tcl commands, which allows you greater functionality. These commands, referred to as *Tcl-equivalent commands*, are registered with the Tcl interpreter using C functions.

You can convert scripts and flows from Scheme to Tcl-equivalent commands, which eliminates Scheme operations from your flow and enables you to use Tcl throughout. The Tcl-equivalent commands have the same names and arguments as Scheme commands; however, because Scheme and Tcl are different syntactically, when you use a Tcl-equivalent command, you must follow the syntax of the Tcl language. For information about converting Scheme commands to Tcl-equivalent commands, see [Table 1-1 on page 1-7](#).

The Milkyway Scheme extension language provides a large set of commands and an easy-to-use GUI, which executes Scheme commands and provides a logging file that can be replayed. Tcl retains these benefits. In addition, the Tcl-equivalent command set facilitates user-shell scripting. However, Tcl-equivalent commands have some important restrictions. It is recommended that your new work flows be developed around Synopsys Tcl, as described in the following section. For more information about Tcl-equivalent restrictions, see [“Tcl-equivalent Command Limitations” on page 1-10](#).

Tcl Support for Tools Based on Milkyway

Milkyway supports common Tcl with two Tcl command styles. Note that there is only one Tcl interpreter, the Common Command Interpreter (CCI), which is used by Design Compiler and Physical Compiler.



Scripts that are written in Synopsys Tcl are the most portable across multiple tools and multiple releases than those written using Tcl-equivalent commands. Synopsys Tcl scripts are also more concise and easier to modify than scripts written in Scheme or with the Tcl-equivalent commands.

Tcl implementation falls into the following categories:

- Common Tcl commands

These basic Tcl commands include `set`, `echo`, `foreach`, and `if` `then` `else`.

- Tcl-equivalent commands of Scheme commands

These are Scheme commands that you convert to Tcl by modifying the Scheme syntax. A guideline for one-to-one mapping from Scheme commands to the Tcl-equivalent commands is provided in [Table 1-1 on page 1-7](#). This mapping makes it easier to convert existing Scheme scripts into Tcl scripts, and, in many cases, only local syntax changes are required to make a script run in Tcl.

- Synopsys Tcl commands

Synopsys Tcl refers to the Tcl language with the Synopsys CCI (Common Command Interpreter) layer. Examples of Synopsys Tcl include the `get_*` commands (for example, `get_cells` and `get_libs`), `open_design`, and `close_design`. To list the Synopsys Tcl commands, type `help` in the Tcl shell. For more information, see [“Introduction to Synopsys Tcl” on page 1-11](#).

All new commands are written in Synopsys Tcl using the Common Command Interpreter. This enables compatibility with other Synopsys tools in the implementation flow, including Primetime, Design Compiler, and Physical Compiler.

Tcl-Equivalent Command Guidelines

Tcl-equivalent commands are provided for Scheme commands that contain at least one uppercase character and no punctuation. For Scheme commands that fit these criteria, you can map between Scheme and Tcl syntax by following the mapping guidelines in [Table 1-1](#).

Table 1-1 Scheme to Tcl Syntax Mapping

Scheme Syntax	Tcl Syntax
#t / #f	1/0 (or true /false)
"string"	string or "string" (double quotation marks are optional)
'(1 2 3)	{1 2 3}
(geGetEditCell)	[geGetEditCell]
define (variable)	set
define (procedure)	proc
load	source
; comment	# comment

Tcl syntax is convenient for working with Tcl-equivalent commands. The space and underscore keys are equivalent for arguments based on the GUI replay commands. Because the Tcl command log uses the underscore, you can remove the double quotation marks. In fact, double quotation marks are optional in Tcl. A sample from a Tcl log might look like the following:

```
geOpenCell
setFormField open_cell cell_name p11
formOK open_cell
```

Scheme and Tcl Syntax and Semantics

There are differences in syntax between Scheme and Tcl-equivalent commands. As shown in [Table 1-1](#), Tcl commands have different syntax for Boolean values and lists. For example, in Scheme you write

```
setDisplayMode "QueryBoxes" #t  
(myDrouteOptContactABC123 (geGetEditCell) '(  
    ("v1" "v1" 2))
```

In Tcl, you write instead

```
setDisplayMode QueryBoxes 1  
myDrouteOptContactABC123 [geGetEditCell] {  
    {"v1" "v1" 2} }
```

The following list highlights the major differences between the two extension languages. Keep these differences in mind as you modify Scheme syntax to create Tcl-equivalent commands. In general, Tcl is different from Scheme in that

- Variables in Tcl must be preceded by a dollar sign (\$) when they are accessed, but not when they are set.
- Setting a variable in Tcl is done with the `set` operation and not `define`.
- Truth is expressed in Tcl with `true` rather than `#t`, and falsehood is expressed with `false` rather than `#f`. (You can also use `1` and `0` in Synopsys Tcl for true and false.)
- Tcl `proc` replaces Scheme functions.
- You `source` a file in Tcl to bring it into the tool instead of loading it.

- Comments in Tcl begin with the pound character (#) and run the length of the line, rather than beginning with a semicolon (;) as in Scheme.
- Commands in Tcl do not need to be surrounded by parentheses as in Scheme.
- Most strings in Tcl do not require quotation marks around them, but most arguments in Scheme do require double quotation marks.
- Generally, Tcl has fewer commands with more arguments, most of which are named parameters and most of which are optional. Scheme has a greater number of commands with fewer arguments, and each argument is positional and required.
- Scheme commands are tied into the GUI, where each menu setting is reflected in a new command line that sets an option, which begins when you issue a `formOK` command. Tcl commands have the options built into the commands and are executed when you enter them on the command line.
- In Tcl, you have a prompt. When you enter a multiline command, the lines that follow do not have a prompt. This helps to confirm when you have entered a complete command and when you have an unbalanced set of parentheses. (In Tcl, if the line ends with an open bracket, it can continue to multiple lines and end with newline characters.)
- To print out messages in Tcl, you can use `echo`.
- If you want to execute a command inside a command in Tcl, you put it in square brackets [].
- Most commands in Tcl are based on the `verb_[modifier_]noun` form as in `report_timing` or `set_false_path`.

- There are no uppercase letters in Tcl commands. However, you do use uppercase letters in Scheme commands.
- You have a current UNIX directory in Tcl that you can change by using the `cd` command, observe by using `pwd`, and explore by using `ls`.

Tcl-equivalent Command Limitations

As mentioned previously, Tcl-equivalent commands based on Scheme syntax are provided for Scheme commands that contain at least one uppercase character and no punctuation. However, unlike Synopsys Tcl commands, there are no man pages available for Tcl-equivalent commands. Instead, see the Physical Implementation Online Help.

Another significant limitation is that Tcl-equivalent commands do not accept collections as arguments, nor do they produce collections. Also, the command abbreviation mode is not supported. (These limitations do not pertain to Synopsys Tcl commands as listed in “[Synopsys Tcl Commands](#)” on page 1-28.)

In addition, some Scheme commands are not provided as Tcl-equivalent commands, such as `cmdGetLastError`, which was replaced with `error_info`. The `clf*`, `RPC*`, `tdf*` and other Scheme commands that are obsolete are not available as Tcl-equivalents. For a list of these commands, see “[Scheme Commands Not Supported in Tcl](#)” on page 1-70.

Introduction to Synopsys Tcl

Tcl capability in Milkyway is complemented by a Common Command Interpreter (CCI), an extension layer with enhanced features on top of Tcl. Synopsys Tcl refers to the Tcl language with the Synopsys CCI layer.

Synopsys Tcl is the Synopsys standard language. It is a common interface for tools based on Milkyway and simplifies using standard Tcl for IC design and verification. One of the main benefits of Synopsys Tcl is the support of Synopsys collections. For information about Synopsys collections, see “[Synopsys Tcl Collections in Milkyway](#)” on page 1-12.

Synopsys Tcl provides consistent, easy-to-use command syntax that follows a verb-noun format, such as

```
report_timing
```

Note that successful redirection depends on the commands used. The commands support keyword-based arguments, such as

```
report_timing -net -from A/Z -to output Reg1/D
```

Synopsys Tcl commands also support command redirection, such as

```
report_timing > my_timing.rpt
```

where the report timing output goes into a file named my_timing.rpt

Synopsys Tcl commands are supported by command man pages and provide an extended Milkyway user message system. To get command help in Tcl, you use the `help` command. You can also use a wildcard (*), for example, `help get_*`.

To see a man page in Tcl, you enter `man command_name` on the command line. For information about man pages, enter `man help` on the command line.

Another benefit of Synopsys Tcl over Scheme is its greater selection and operation capability. For more information about using Tcl, collections, and usage with Synopsys tools, see the *Using Tcl With Synopsys Tools* manual.

Synopsys Tcl Collections in Milkyway

This section covers some highlights about using collections in the Milkyway environment. It is not intended to be a complete discussion about collections. For more information, see the collections man page and *Using Tcl With Synopsys Tools*.

Collections in Synopsys Tcl are groups of objects. They can be assigned to variables and passed to arguments and commands. They are referenced by object ID and name. Collections allow you to process groups of objects as easily as a single object. Working with collections involves commands that produce collections and commands that consume collections. Many commands that produce collections begin with `get_`, such as

```
set inputs [get_ports -filter "direction==input" J*7]
```

which finds all the ports that match the expression `J*7` and then filters them by direction. In this case, ports that are not of direction `input` are excluded. The remaining ports, if any, are stored in the Tcl variable `inputs` as a collection.

Collections are efficient internal representations of design objects. They are not just strings or Tcl lists. They describe the class of the objects that comprise them: a collection with a port NN is completely distinct from a collection with a cell NN.

A partial list of supported logical objects includes port, cell, pin, and net. Examples of physical objects are bound, terminal, net_shape, route_guide, and placement_blockage. Finally, objects can be timing related, such as clock.

```
set net_clct [get_nets A/B/C/*]
    remove_nets $net_clct

query_objects -verbose [get_selection]
```

In the previous example, note that objects can be from the design (cell), from the library (lib_cell), or only present at runtime (clock). Also note that Milkyway Synopsys Tcl objects are not strings. They define the type of the object and are automatically managed as the design is updated.

With collections you can select the objects that you need. You can then pass these objects to elaborate operations such as device sizing or area I/O support.

A sample of basic collection operations follows:

```
Set p [get_ports -filter "direction==input" *n*]
{n1 n2 in1 in2 in3 bnnn}

Set i [sort_collection $p name]
```

```

{bnnn in1 in2 in3 n1 n2}

foreach_in_collection nn $i {
    echo [get_att $nn full_name] \
        [get_att $nn area]
}

```

In summary, for most applications you can create Synopsys Tcl scripts that use a combination of collections starting with `get_*` operations. You then filter, sort, and operate on these objects.

First Class Objects

Technically, collections are internal data structures representing ordered lists of first class objects. Generally, a first class object is one that

- Has a name
- Has attributes (accessible with `get_attribute`)
- Can appear in collections
- Does not need to be persistent

Attribute Support

Attributes are central to Tcl because they form a convenient way to access information without introducing new commands. The following is a short introduction to Milkyway attribute support.

All attributes can be accessed with the `get_attribute` command. In addition, attributes can be used in the `-filter` option of all the `get_object` and `sort_collection` commands. To generate a complete and current list of attributes, use the `list_attributes -application` command.

The following is a list of common attributes:

`name`

Note that some physical objects have names that are just their object ID with an alphabetic prefix.

`object_class`

Available on all objects, such as net and cell.

`object_id`

These are relevant only to Milkyway objects that have specific object IDs. These identifiers can be useful in a combination with Synopsys Tcl, Tcl-equivalents, and Scheme scripts that accept object IDs.

`bbox`

The bounding box of the object (physical objects only, not nets or clocks).

Milkyway provides application attribute support with hundreds of supported attributes. Examples of attributes include `full_name`, `object_type`, `ref_name`, and `attached_files`.

Milkyway also supports attribute manipulations, such as

```
list_attribute -application  
get_attribute [get_cell buf101] area
```

Using attributes, you can write reports, such as a list of pins with specific characteristics of interest. You can also use attributes for collection filtering, such as

```
get_cells "mycell*" -filter "@ref_name==BUF1"
```

Synopsys Tcl Help, Man Pages, and User Message Handling

The Synopsys Tcl commands are supported by man pages and enhanced Milkyway error message handling. (Note that Scheme commands and Tcl-equivalent commands based on Scheme are not supported by man pages; however, they are documented in Physical Implementation Online Help and in the PDF manuals.)

Synopsys Tcl Help

Synopsys Tcl Help is a quick resource for command information. The following basic commands can help get you started with Synopsys Tcl Help:

To find a listing of Milkyway and collections operations enter the following at the Milkyway command line:

```
Milkyway> help
```

To find tool commands with a description that are based on Milkyway enter

```
Milkyway> help milkyway
```

To find a description of a specific command enter

```
Milkyway> help command_name
```

To find detailed information about commands and arguments enter

```
Milkyway> help -verbose command_name
```

To find a detailed list of Milkyway Synopsys Tcl commands and arguments enter

```
Milkyway> help -verbose milkyway
```

To find a list of attribute operations enter

```
Milkyway> help *attr*
```

Man Page Support

The man pages provide detailed information about Synopsys Tcl commands, including the syntax, arguments, description, and examples. For more information about the man pages, enter `man help` on the command line or enter `man command_name` for information about a specific Tcl command.

Milkyway provides man pages for error messages in a Synopsys Tcl shell. These man pages provide details about the error and provide information about how to correct the problem. Each message has a message ID, such as MWUI-001, and a corresponding man page. For example, you might encounter the following message:

```
Error: 'aaa' is not an open library (MWUI-001)
```

In Milkyway Tcl mode, you can find more details about the message by entering the following on the command line:

```
Milkyway> man MWUI-001
```

This opens a man page, as shown in [Figure 1-2](#).

Figure 1-2 Example of a Man Page Error Message

```
messages(n)                                     messages(n)
                                                Dec, 2004

NAME
    MW-001 (error) '%s' is not an open library.

DESCRIPTION
    The library you specified is not open in current session.

WHAT NEXT
    If the command cause this error is close_mw_lib, nothing need to do.
    The library is closed already. Or else, open the library use
    open_mw_lib, then try again.

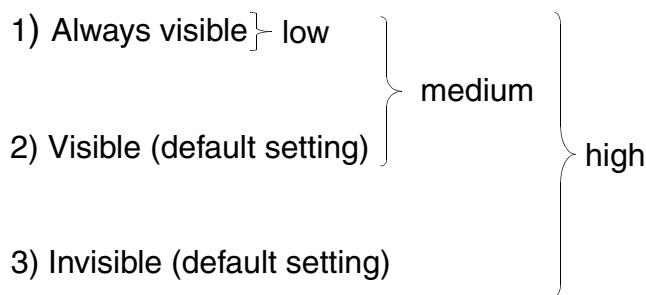
SEE ALSO
    close_mw_lib(2), open_mw_lib(2).
```

User messages are also available in the man pages. Other Tcl commands for messages include `suppress_message`, `unsuppress_message`, and `print_suppressed_messages`.

Enhanced Milkyway Message Handling

The enhanced message handling system provides commands that allow you to include or exclude certain predefined message categories so you can focus on the messages that are important and relevant to you. There are three groups of messages based on their level of visibility, as shown in [Figure 1-3](#).

Figure 1-3 Message Visibility Settings



Using the `dbSetMsgLevel` command, you can set the message level to low, medium, or high. The default visibility setting is medium. As shown in [Figure 1-3](#), setting the message level to high displays the messages that are invisible by default. Conversely, setting `dbSetMsgLevel` to low reduces the number of messages that are visible.

You can use the `dbGetMsgLevel` command to display the current message level. For more information about the `dbSetMsgLevel` and `dbGetMsgLevel` commands, see [Physical Implementation Online Help](#).

Scheme and Tcl Execution Modes

The usage of Scheme and Tcl modes is similar. From the Milkyway GUI, you can choose between Scheme mode and Tcl mode by clicking either the Scheme button or the Tcl button at the bottom-left corner of the GUI.

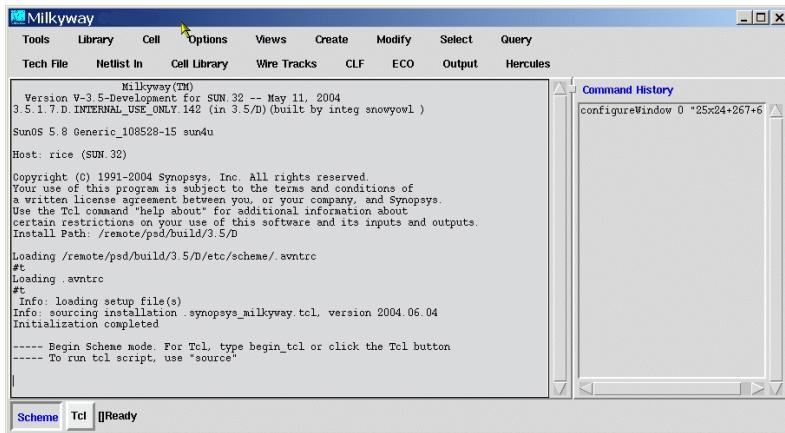
Using Scheme Mode

To invoke Milkyway at a UNIX prompt, enter

```
% Milkyway &
```

There is no prompt in the Milkyway GUI command window, as shown in [Figure 1-4](#).

Figure 1-4 Milkyway GUI Using Scheme



The Milkyway GUI menus invoke Scheme commands. You can map bind-keys, strokes, and mouse buttons to Scheme functions. In Scheme mode, the replay file is named `my_file.cmd.date-time`.

Accessing Tcl From Scheme

If you need to execute a Tcl command in Scheme mode, click the Tcl button in the lower-left corner of the Milkyway GUI, or enter

```
'(tcl "my-tcl-command arg1 arg2")
```

To load a Tcl script, enter

```
source "my_file.tcl.date-time"
```

Note that the `tclLoad` and `loadUserForm` commands are still supported for backward compatibility.

To get a Tcl variable, enter

```
getTclVar
```

Note that the `userGetTclVar` command is still supported for backward compatibility.

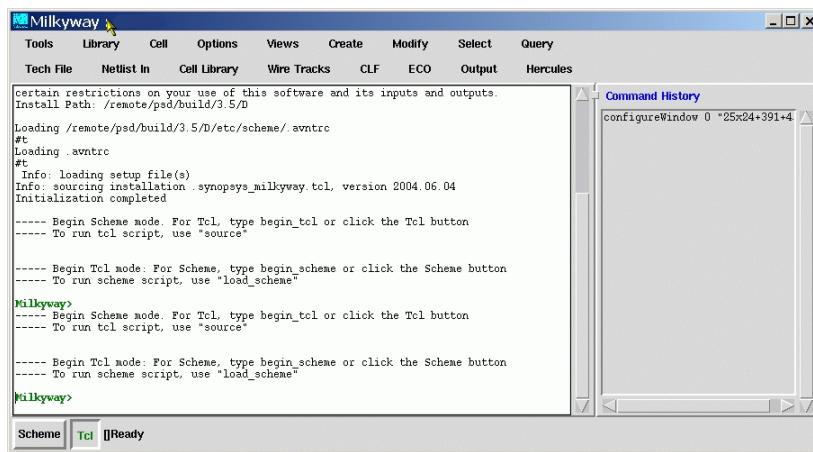
Using Tcl Mode

The Tcl shell works similarly to the Scheme shell. To invoke Milkyway at the UNIX prompt, enter

```
% Milkyway -tcl &
```

In Tcl mode, there is a prompt in the Milkyway GUI command window. The default is a white background with a green prompt, as shown in [Figure 1-5](#).

Figure 1-5 Milkyway GUI Using Tcl



The Milkyway GUI menus invoke Tcl functions. You can also map bind-keys, strokes, and mouse buttons to Tcl functions. In Tcl mode, the replay file is named ***tcl***, as in **my_file.tcl.date-time**, instead of ***cmd*** as it is in Scheme.

Accessing Scheme From Tcl

If you need to execute a Scheme command in Tcl mode, click the Scheme button in the lower-left corner of the GUI, or enter the following on the command line:

```
Milkyway> scheme {my-scheme-command "argument"}
```

Note that the `userEvalScm` command is still supported.

To load a Scheme script enter

```
Milkyway> load_scheme "my_file.cmd.date-time"
```

To get a Scheme variable, enter

```
Milkyway> get_scheme_var
```

Note that the `userGetScmVar` command is still supported.

Milkyway Synopsys Tcl provides an `object_id` attribute. This allows a Tcl script to invoke Scheme commands that require an object ID.

Milkyway Library Search Management

Beginning with the W-2004.12 release, Milkyway library preparation methodology has been enhanced to support the `search_path` and `link_library` variables used by Design Compiler and Physical Compiler. Milkyway also provides new library search variables that specify logic model .db files.

The following variables determine the logical library search order:

search_path
link_library } from Design Compiler and Physical Compiler

mw_min_library
mw_max_library
mw_typ_library } new, only for Astro family tools

Note:

The library search methodology described in this section is separate from library preparation using the `gePrepLibs` command. The preceding search variables have precedence over any design-specific logic model .db file settings in the design library and any default logic model .db file settings in the reference libraries that are set with the `gePrepLibs` command.

The physical library search order is maintained separately by the Milkyway design library and reference control file commands. For information about specifying reference library settings, see [“Reference Library Management” on page 1-27](#).

Setting `search_path` and `link_library`

There are two ways to set the search path and link library in Milkyway, Astro, or Jupiter.

In Tcl mode, enter the `set search_path` and `set link_library` commands at the Milkyway prompt.

```
Milkyway> set search_path "search_path ./unmapped"
```

```
Milkyway> set link_library {core_slow.db}
```

Alternatively, you can set these variable setting in a Tcl file and source that file in the Milkyway GUI in Tcl mode. For example, you can create a `.synopsys_dc.setup` file with these commands:

```
set link_library {~/special_file.db}
set search_path "search_path ~/special_location"
```

At the Milkyway prompt, you can source this Tcl file by entering

```
Milkyway> source .synopsys_dc.setup
```

To check the values of these commands you can use the `echo` command:

```
Milkyway> echo $link_library
~/special_file.db
```

```
Milkyway> echo $search_path
~/special_file.db
```

Design Compiler, Physical Compiler, and PrimeTime do not pass the values of the link library or search path to Milkyway. You must enter them with the `set` command or source a Tcl command file.

Logic Library Search Order Variables

The `search_path` variable defines the list of paths used to locate the logic `.db` files. It is a space-delimited list of directory names used to resolve `.db` file names.

Note that if the directory is a Milkyway library, the directory path must include the required view directory, for example, /home/johndoe/milkyway_std_lib/LM). If a .db file name does not include a directory path, Milkyway attempts to locate it in each directory in the search_path list, until the file is located.

The link_library, mw_min_library, mw_max_library, and mw_typ_library library search variables are ordered, space-delimited lists of .db file names (with or without directory paths). The .db files are used for the following:

- link_library
 - Used for all operating conditions at the end of the logic library search order.
- mw_min_library
 - Used for the minimum operating condition.
- mw_max_library
 - Used for the maximum operating condition.
- mw_typ_library
 - Used for the typical operating condition.

You must define search_path and one of the library variables (link_library, mw_min_library, mw_max_library, or mw_typ_library) for the search variables to take effect. When you do this, the search variables are used whether or not all the library variables are defined.

During runtime, the search variables and their corresponding values are stored in your user environment. In determining the logical library search order, Milkyway uses only the current set of values for all designs in the current Milkyway session. The set can be modified by successive `set` commands or `source` commands.

When a cell is saved, the search variable values are saved to the cell, overwriting any previously saved values. This allows applications to use the same search variables for the same cell.

Note that Milkyway cannot automatically restore the saved values. The tool because Milkyway does not know which search variable values to use when multiple cells are opened, because each cell contains different search variable values.

In the Milkyway GUI, you can define and set the search variables in a Synopsys Tcl setup file. You can also directly call the `set` command in Milkyway Tcl mode to set search variables without using a setup file.

Ordered .db File List

When your tool retrieves the logic model .db files, Milkyway uses the current search variables to create an ordered .db file list. If the variables do not exist, the logic model .db files are obtained through the `gePrepLibs` settings.

Starting with the first .db file in each list, each `mw_min_library`, `mw_max_library`, and `mw_typ_library` .db file is resolved against the `search_path` directories (if necessary) and added to its respective minimum, maximum, or typical .db file list.

Next, each `link_library .db` file is resolved against the `search_path` directories (if necessary) and added to the end of the minimum, maximum, and typical .db file lists. To resolve a .db file, Milkyway attempts to locate it in each `search_path` directory, starting with the first directory, until the file is located.

Reference Library Management

For backward compatibility, you can continue to specify reference library settings through the GUI or by loading a reference control file. Milkyway uses the reference control file settings by default. You can switch the default to GUI by using the `dbSetLibRefCtrl FileMode` command.

Milkyway maintains consistency between GUI settings and the reference control file. Because the GUI covers only the top library's references, only the top-level references are consistent.

If you modify the reference control file, the modified top-level references overwrite the internal GUI settings. Whenever you modify the GUI settings, the modifications incrementally apply to the reference control file.

You can continue to use the `gePrepLibs` command to import and set logic model .db files for library preparation. However, the `gePrepLibs` logic model settings are secondary to the logical library search order described earlier. You should use the `gePrepLibs` logic model settings only if you do not set the search variables.

Synopsys Tcl Commands

This section provides an overview of the Synopsys Tcl commands. For more information, see the man page for the command of interest.

As shown in [Table 1-2 on page 1-30](#), a single Synopsys Tcl command often replaces one or more Scheme commands. There are also Synopsys Tcl commands that provide new functionality and do not replace Scheme commands. For example, the `get_*` commands are unique to Synopsys Tcl.

Synopsys Tcl commands fall into the following categories:

- Milkyway object access commands, such as `open`, `create`, and `get`
- Logical and physical editing commands, such as `connect`, `move`, and `rotate`
- Attribute queries, such as `set`, `list`, and `remove`
- Utilities commands, such as `report` and `update`

You can find more information about the following Synopsys Tcl commands by entering `help milkyway` on the command line:

```
add_reference_library # Add reference libraries
all_connected        # Report all connection to an object
attach_file          # Attaches a file to a library/design
change_selection     # select the specified objects, and
                     # highlight on the graphic window
close_design         # Closes the design
close_mw_lib         # Close milkyway libraries
connect_net          # Connect a net with port instance/port
copy_design          # copy design(s) from one library to
                     # another
copy_mw_lib          # Copy a milkyway library
create_bound         # Creates a new instance bound
create_cell          # creates cell
```

```

create_design      # creates a new design
create_mw_lib      # Create a milkyway library
create_net         # Create nets in design
create_net_shape   # Creates net shape
create_physical_port # creates a physical port for a specified port
create_placement_blockage # creates one blockage for placement in a special region
create_port        # creates a port for a design
create_route_guide # creates one blockage for routing on
                   # a special region
create_route_shape # Creates route shapes for a net
current_design     # Specifies a design you want to set as the working design
current_mw_lib      # Get current milkyway library
define_user_attribute # Define a new attribute
detach_file        # Detaches a file from a library/design
disconnect_net     # Disconnect a net with its members
get_attached_file  # Gets the real path from the pseudonym
get_bounding_box   # find the bounding box of a group of objects
get_bounds          # Creates a collection of bound
get_layers          # get available layers in the current library
get_selection       # get objects that were selected graphically
get_via_masters    # get via masters in the current library
move_objects        # Moves one or more objects
open_design         # Opens the design
open_mw_lib          # Open a milkyway library
remove_attribute    # Remove attribute on the specified object
remove_bounds       # Removes a list of bounds
remove_cell          # removes cells
remove_design        # removes design(s) from the current library
remove_net          # Remove nets in design
remove_net_shape    # Removes net shapes
remove_object        # remove objects from design
remove_physical_port # Removes a collection of physical ports
remove_placement_blockage # Removes a collection of placement blockages
remove_port          # removes ports

```

Table 1-2 maps Scheme commands to new Synopsys Tcl replacement commands. The table also includes Synopsys Tcl commands that provide new functionality and that do not replace Scheme commands.

Table 1-2 Mapping Scheme to Synopsys Tcl Commands

Scheme commands	Synopsys Tcl commands and syntax
none	about # Prints a message about the copyright.
dbAddRefLib libId “refLibName”	add_reference_library # Adds reference libraries.
cmRefLib	[-to mw_lib] (Main library to add to) [-lib_id lib_id] (ID of the main library: Value >= 1) libraries (List of reference libraries to add)
none	add_to_collection # Adds objects to a collection. The result is a new collection. [-unique] (Remove duplicates from the result) collection1 (Base collection) object_spec (Objects to add)
dbGetConnectedObject cellId objId	all_connected # Reports all connections to an object. object (Object of which connections are returned)
dbStoreCellAttachedFile cellId “attachedFileName” “pseudoName” “comment”	attach_file # Attaches a file to a library/design. [-class class name] (Class of the object) [-to object name] (Object to attach the file) [-comment string] (Comment associated with the attached file) [-pseudonym string] (Pseudo-name for the attachee) file_name (File to be attached)
dbAddLibSchemeFile libId “supportFileName”	

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
none	<p>begin_scheme</p> <p>When running in Tcl mode, you have the option of switching to Scheme mode by issuing the “begin_scheme” command. Once in Scheme mode, you can switch back with the “begin_tcl” command. Use the “scheme” command for individual Scheme commands.</p> <p>See Also: scheme</p>
none	<p>change_selection # Selects the specified objects and highlights on the graphic window.</p> <p>[-replace] (Replace current selection; its default) [-add] (Add to current selection) [-remove] (Remove from current selection) [collections] (Collections of objects to be selected)</p>
none	<p>check_mw_lib # Checks Milkyway libraries.</p> <p>[-technology] (Checks the technology information) [-design_drc] (Checks the cell DRC) [-treat_blockage_as_thin_wire] (Treats the blockage as thin wire) [-lib_id lib_id] (Specifies the ID of the library to check: Value >= 1) [libraries] (Specifies the list of libraries to check)</p>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
dbSaveCell cellId dbCloseCell cellId dbAbortCell cellId dbRollBackCell cellId	close_design # Closes the design. [-discard] (Discards the change on the design) [-verbose] (Be verbose) [design list] (Design to be closed)
dbCloseLib [libId]	close_mw_lib # Closes Milkyway libraries. [-lib_id lib_id] (ID of library to close: Value >= 1) [mw_libs] (List of libraries to close)
none	collection_to_list # Formats collection contents as a Tcl list or Scheme string. [-name_only] (Do not display type of each object) [-no_braces] (Do not enclose results in braces {}) [-brace_with_quotes] (Use quotation marks “ “ to enclose results, not braces {}) [-newline] (Use newlines to separate names in output) [-truncate elem_count] (Truncate display to n elements) obj_spec_list (Objects to find and display)
none	compare_collections # Returns 0 if two collections contain the same object. [-order_dependent] (Order of objects matters) collection1 (Base collection) collection2 (Compare this to collection1)
dbConnect cellId netId memberId	connect_net # Connects a net with port instance/port. net name (Name of the net to connect) object list (List of objects to connect)

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
none	<pre>copy_collection # Makes a copy of a collection. The result is a new collection. collection1 (Collection to copy)</pre>
dbCopyCell "fromCell" "toCell"	<pre>copy_design # Copies designs from one library to another.</pre>
dbCopyCellFromAnother Lib "fromCell" "fromLib" "toCell"	<pre>-from source_design (Designs to be copied, default view is CEL) [-to target_design] (Design to be created; default is the same name) [-from_library source_library]</pre>
dbCopyCellToAnotherLib Disk "fromCell" "toLib" "toCell"	<pre>(Library from which to copy design; default is current library) [-to_library target_library] (Library to which to copy design; default is current library) [-with_tim] (Copy the TIM view too)</pre>
dbCopyLib "fromLib" "toLib"	<pre>copy_mw_lib # Copies a Milkyway library. [-from mw_lib] (Specifies the source library to copy) [-from_lib_id lib_id] (ID of the source library to copy: Value >= 1) -to lib_name (Name of the destination library)</pre>
none	<pre>cputime # Specifies the CPU time used. [format] (Includes microseconds and prints in a given print format)</pre>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
dbCreateCellRegion cellId pointArray	<pre>create_bound # Creates a new instance bound.</pre>
dbCreatePlanGroup cellId planGroupName groupType targetPercent lIX lIY urX urY	<pre>[-auto] (Attempts to automatically create the bound) [-bounding_box list] (A rectangle to define the geometry of the bound) [-poly list] (The list of points to form the polygon) [-dimension list] (Specifies the dimension of a groupbound) [-effort one-of_string] (Specifies the effort to bring cells closer inside a groupbound. Default: medium. Values: low, medium, high, ultra) [-name string] (The name of the bound) [-logical_name string] (The name of attached logical node) [-attached_instance list] (The list of attached instance) [-utilization float] (The target utilization; default is 0.7: Range: 0.1 to 1) [-type type] (Type of the bound: "soft" or "hard"; default is "soft") [-color string] (Color of the bound: string or index) [-rigidity int] (Rigidity of cells inside the bound; default is 10: Range: 1 to 10) [-exclusive] (Creates an exclusive bound) [-fixed] (Creates a bound with fixed location) [list] (Specifies a list of cells to be included in the groupbound)</pre>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
dbCreateCellInst cellId “childLibName” “childCellName” “cellInstName” “rotationStr” “mirrorStr” Points [“topCellName”]	create_cell # Creates a cell. -from_design design_name (Design from which to create the cell) [-from_library library_name] (library from which to create the cell) [-rotation 90 180 270] (Degree to which you want the cell rotated; default: 0) [-mirror x y] (Orientation of the cell; default: 0) -origin origin (Reference point for the cell) cell_name (Name of cell to be created)
dbCreateCell “cellName” “viewName” “viewType”	create_design # Creates a new design. [-view {cel netl fram exp tim}] (View name of the design; default: cel) [-view_type {layout netlist schematic symbolic unknown}] (View type of the design; default: layout) [-verbose] (Be verbose) design_name (Name of the design to be created)

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
dbCreateLib "libName" "techfile" ["hierSep"]	<pre>create_mw_lib # Creates a Milkyway library.</pre> <code>[-hier_separator sep]</code> (Hierarchical separator; default is "/") <code>[-case_sensitive]</code> (Create case-sensitive library) <code>-technology file_name</code> (Technology file name) <code>lib_name</code> (Name of library to create)
dbCreateChildNet cellId "childNetName" netId ["topCellName"]	<pre>create_net # Creates nets in design.</pre>
dbCreateHierNet cellId "netName" netId cellRegionNumber ["topCellName"]	<code>[-design design name]</code> (Create a net in specified design) <code>net names</code> (Names of nets to create)
dbCreateNet cellId "netName"	

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
dbCreateHorizontalWire cellId masterWireId wireLength xPoint yPoint	create_net_shape # Creates wire shapes for a net. -bounding_box {{ll_x ll_y} {ur_x ur_y}} (Coordinates of lower-left and upper-right corners)
dbCreateVerticalWire cellId masterWireId wireLength xPoint yPoint	-origin {x y} (The point where you place the net shape) [-length length] (The length of the wire net shape) [-width width] (The width of the wire master)
dbCreateWireMaster cellId pathType layerType wireWidth layerNumber	[-path_type {0 2}] (The alignment type of the wire edge; value: 0 (square, no extension), 2 (square with half width extension)) [-layer layer] (The layer on which to create wire net shape) [-net net] (The net on which to create wire shape) [-vertical] (Indicates that the wire is vertical; default is horizontal.)
dbMakeWireMasterId cellId [layer] internWireMasterId	
none	create_placement_blockage # Creates one blockage for placement in a special region. -bounding_box {{ll_x ll_y} {ur_x ur_y}} (Coordinates of lower-left and upper-right corners) [-type { hard soft }] (The type of blockage; default is hard)

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
<pre>dbCreateChipPort libName cellName netName portName</pre> <pre>dbCreatePort cellId netId "portName" [“topCellName”]</pre>	<pre>create_port # Creates a port for a design.</pre> <pre>-net net (The net on which to create port) [-direction {input (default) output tristate}] (Direction of the created port) port_name (Name of port to be created)</pre>
<pre>dbCreateRectangle cellId “layerName” layerNumber pointArray</pre>	<pre>create_route_guide # Creates one blockage for routing for wiring in a special region.</pre> <pre>-bounding_box {{ll_x ll_y} {ur_x ur_y}} (Coordinates of lower-left and upper-right corners) [-layer layer] (Layer for the wiring keepout)</pre>
<pre>dbCreateContact cellId contactNumber point</pre>	<pre>create_route_shape # Creates route shapes for a net.</pre> <pre>-at {x y} (The point where you place the route shape) [-master contact_name] (The via master's name defined in the library's technology file) [-net net] (The net on which to create route shape)</pre>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
dbCreatePin cellId netId “pinName” “layerName” pointArray [“topCellName”]	<pre>create_terminal # Creates a pin record for a specified port.</pre> <p> [-name name] (Name for the port shape) -port port (Name of the port to which the port shape is added) -layer layer (Layer for the port shape) -bounding_box {{ll_x ll_y} {ur_x ur_y}} (Coordinates of lower-left and upper-right corners) </p>
none	<pre>create_text # Creates a new instance text.</pre> <p> [-layer default value indicated by a gvar mw_text_layer] (Specifies the layer on which to create the text) [-height default value indicated by a gvar mw_text_height] (Specifies the height indicated for the text) [-orient default is 0] (Specifies the orientation you want for placement of the text) [-anchor one-of-string] (The point in the text line that you want placed at the point you specify. Default: bl) -origin {x y} (Specifies the point where you place the net shape) string (Specifies the text name you want to add) </p>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
dbCreateVoltageArea topCellId voltageAreaName priority guardband colorCode PointArray	<pre>create_voltage_area # Creates a new instance voltage area.</pre> <p> [-color default value indicated by a gvar mw_va_color] (Specifies the color for the voltage area) </p> <p> [-bounding_box list] (Specifies the rectangle to define the geometry of the bound) </p> <p> [-poly list] (Specifies the list of points to form the polygon) </p> <p> [-guardband default value indicated by a gvar mw_va_guardband] (Specifies the space along the boundary of the voltage area where cells cannot be placed) </p> <p> [-priority default value indicated by a gvar mw_va_priority] (Specifies the the color for the voltage area) </p> <p> [-cells list] (Specifies the list of cells associated with the voltage area) </p> <p>string (Specifies the voltage area name you want to create)</p>
geGetEditCell	<pre>current_design #Specifies a design you want to set as the working design.</pre> <p>[design_name] (Design name to be set as current)</p>
none	<pre>current_instance # Sets or gets the current instance.</pre> <p>[Instance object or name] (Specifies the design name)</p>
dbGetCurrentLibId	<pre>current_mw_lib # Gets current Milkyway library.</pre>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
<pre>dbCreateCellProperty cellId "propName" propValue</pre> <pre>dbCreateProperty cellId objectId "propName" value</pre>	<pre>define_user_attribute # Defines a new attribute.</pre> <pre>-type one-of-string (Datatype for the newly defined user attribute)</pre> <pre>-class list (Class type for the object on which to define the attribute)</pre> <pre>[-range_min double] (Min value for numeric ranges)</pre> <pre>[-range_max double] (Max value for numeric ranges)</pre> <pre>[-one_of list] (List of allowable strings)</pre> <pre>[-quiet] (Do not report any messages)</pre> <pre>attr_name (Name of the attribute)</pre> <pre>attr_name</pre>
<pre>dbDeleteCellAttachedFile cellId "pseudoName"</pre> <pre>dbDeleteLibAttachedFile libId "pseudoName"</pre>	<pre>detach_file # Detaches a file from a library/design.</pre> <pre>[-class class name] (Class of the object) [-from object name] (Object to detach the file) pseudo name (Pseudo-name to be detached)</pre>
<pre>dbDisconnect cellId netId memberId</pre>	<pre>disconnect_net # Disconnects a net with its members.</pre> <pre>-all (To disconnect all members) net name (Name of net to disconnect) object list (List of objects to disconnect)</pre>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
none	<pre>dump_tlu_plus_file # Dumps the TLUPlus information for the specified library. -output tlu_plus_file (Dumps the TLUPlus information to this file) [-library mw_libs] (Specifies the list of libraries to dump the TLUPlus information) [-lib_id lib_id] (Specifies the ID of the library to dump the TLUPlus information: Value >= 1)</pre>
none	<pre>exec_and_log Normal Tcl commands entered in the console window, directly, are executed and logged in the log file. However, if you modify the GUI menus to invoke Tcl commands, they will not be logged unless you put the “exec_and_log” command in front of your Tcl command, on the same line. This should be used for your GUI button if the GUI button will trigger a Tcl command.</pre>
none	<pre>filter_collection # Filters a collection, resulting in new collection. [-regexp] (Operators =~ and !~ use regular expressions) [-nocase] (With -regexp, case insensitive match) collection1 (Collection to filter) expression (Filter expression)</pre>
none	<pre>foreach_in_collection # Iterates over a collection. itr_var (Iterator variable) collections (Collections over which to iterate) body (Script to execute per iteration)</pre>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
none	<pre>get_attached_file # Gets the real path from the pseudonym.</pre> <pre>[-class class name] (Class of the object) [-on object name] (Object on which the attached files returned) pseudo name (Pseudo-name to be detached)</pre>
dbGetLayerNumberFrom Name libId "layerName"	<pre>get_attribute # Gets the value of an attribute.</pre>
dbGetLayerNameFrom Number libId layerNumber	<pre>[-class class_name] (If object_spec is a name, this is its class: Values: port, cell, pin, net, design, lib, lib_cell, lib_pin, terminal, route_guide, placement_blockage, net_shape, route_shape, group, bound, clock)</pre>
dbFetchLayerIdList libId	<pre>[-quiet] (Do not report any messages)</pre>
dbGetRegionPort DirectioncellIdregion PortId	<pre>object_spec (Object (single) from which to get the attribute)</pre>
dbFetchLayerInfo libId layerIndex "layerTechType"	<pre>attr_name (Name of the attribute)</pre>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
dbFetchObject libraryOrCellId subObjectOrNil	get_attribute (<i>continued</i>)
dbFetchPinInstByPortInst cellId portInstId	
dbFetchTechInfo libId “techType”	
dbDisplayCellRail Property cellId	
dbFetchTechInfo libId “techType”	
dbFetchContactGeom cellId objId	
none	<p>get_bounding_box # Finds the bounding box of a group of objects.</p> <p>[<i>-offset offset</i>] (Offset the result by this) [<i>-expand factor</i>] (Expand, or reduce, by this factor) <i>obj_spec_list</i> (Objects to find and display)</p>
dbGetRegionIdByCellInst Id cellId cellInstId	<p>get_bounds # Creates a collection of bounds.</p> <p>[<i>-quiet</i>] (Suppress all messages) [<i>-within rectangle</i>] (Bounds must be within the specified rectangle) [<i>-filter expression</i>] (Filter collection with ‘expression’) <i>patterns</i> (Match bound names against patterns)</p>
dbGetPlanGroupByName cellId planGroupName	

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
dbGetCellInstByName cellId “cellInstName”	<pre>get_cells # Creates a collection of cells. [-hierarchical] (Search level-by-level in current instance) [-filter expression] (Filter collection with 'expression') -of_objects objects (Get cells of these pins or nets) [-quiet] (Suppress all messages) [-regexp] (Patterns are full regular expressions) [-nocase] (With -regexp, matches are case-insensitive) [-exact] (Wildcards are considered as plain characters) -object_id integer (Get cell using its Milkyway object ID; defaults to work in current_design) [-design_id integer] (Milkyway ID of design owning this cell) [-hsc separator] (Separator character (default is "/"): Values: /, @, ^, #, .,) patterns (Match cell names against patterns)</pre>
none	<pre>get_designs # Creates a collection of designs. [-hierarchical] (Search level-by-level in current instance) [-filter expression] (Filter collection with 'expression') [-quiet] (Suppress all messages) [-regexp] (Patterns are full regular expressions) [-nocase] (With -regexp, matches are case-insensitive) [-exact] (Wildcards are considered as plain characters) patterns (Match design names against patterns)</pre>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
dbFetchLayerIdList libId	<pre>get_layers # Gets available layers in the current library. [-include_system_layers] (Matches system layers too) [-for_wiring] (Matches wiring layers only) patterns (Matches layer names against patterns)</pre>
none	<pre>get_lib_cells # Creates a collection of library cells. [-filter expression] (Filter collection with 'expression') -of_objects objects (Get lib cells of these cells or lib pins) [-quiet] (Suppress all messages) [-regexp] (Patterns are full regular expressions) [-nocase] (With -regexp, matches are case-insensitive) [-exact] (Wildcards are considered as plain characters) [-hsc separator] (Separator character (default is "/"): Values: /, @, ^, #, .,) patterns (Match lib cell names against patterns)</pre>
none	<pre>get_lib_pins # Creates a collection of library cell pins. [-filter expression] (Filter collection with 'expression') -of_objects objects (Get lib pins of these pins or lib cells) [-quiet] (Suppress all messages) [-regexp] (Patterns are full regular expressions) [-nocase] (With -regexp, matches are case-insensitive) [-exact] (Wildcards are considered as plain characters) [-hsc separator] (Separator character (default is "/"): Values: /, @, ^, #, .,) patterns (Match lib pin names against patterns)</pre>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
dbGetRefLib libName	<pre>get_libs # Creates a collection of libraries. [-filter expression] (Filter collection with 'expression') -of_objects objects (Get library of these lib cells) [-quiet] (Suppress all messages) [-regexp] (Patterns are full regular expressions) [-nocase] (With -regexp, matches are case-insensitive) [-exact] (Wildcards are considered as plain characters) patterns (Match library names against patterns)</pre>
none	<pre>get_net_shapes # Creates a collection of net shapes. [-within rectangle] (Match reserved areas within specified rectangle) [-region region] (Match net shapes intersect specified rectangle) [-at at_point] (Match net shapes overlap the specified point) [-filter expression] (Filter collection with 'expression') -of_objects net_list (Get net shapes of these nets) [-quiet] (Suppress all messages) patterns (Match via, wire net shapes)</pre>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
dbGetNetByName cellId “netName”	<pre>get_nets # Creates a collection of nets. [-hierarchical] (Search level-by-level in current instance) [-filter expression (Filter collection with 'expression') -of_objects objects (Get nets of these pins or ports or cells) [-quiet] (Suppress all messages) [-regexp] (Patterns are full regular expressions) [-nocase] (With -regexp, matches are case-insensitive) [-exact] (Wildcards are considered as plain characters) -object_id integer (Get net using its Milkyway object ID; defaults to work in current_design) [-design_id integer] (Milkyway ID of design owning this net) patterns (Match net names against patterns)</pre>
none	<pre>get_objects_from_ids # Creates a collection from a list of object IDs objectIds (Specifies the list of object IDs)</pre>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
none	<pre>get_pins # Creates a collection of pins. [-hierarchical] (Search level-by-level in current instance) [-filter expression] (Filter collection with 'expression') -of_objects objects (Get pins of these cells or nets) -object_id integer (Get pin using its Milkyway PortInst object id, defaults to work in current_design) [-quiet] (Suppress all messages) [-regexp] (Patterns are full regular expressions) [-nocase] (With -regexp, matches are case-insensitive) [-exact] (Wildcards are considered as plain characters) [-leaf] (Get leaf/global pins of nets with -of_objects) [-hsc separator] (Separator character (default is "/"): Values: /, @, ^, #, .,) patterns (Match pin names against patterns)</pre>
none	<pre>get_placement_blockages # Creates a collection of placement blockages. [-within rectangle] (Match reserved areas within specified rectangle) [-touching rectangle] (Match reserved areas touching specified rectangle) [-filter expression] (Filter collection with 'expression') [-quiet] (Suppress all messages) patterns (Match keepouts)</pre>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
dbGetPortByName cellId "portName"	<pre>get_ports # Creates a collection of ports. [-filter expression] (Filter collection with 'expression') -of_objects objects (Get ports of these nets) -object_id integer (Get port using its Milkyway object ID, defaults to work in current_design) [-quiet] (Suppress all messages) [-regexp] (Patterns are full regular expressions) [-nocase] (With -regexp, matches are case-insensitive) [-exact] (Wildcards are considered as plain characters) patterns (Match port names against patterns)</pre>
none	<pre>get_route_guides # Creates a collection of route guides. [-within rectangle] (Match reserved areas within specified rectangle) [-touching rectangle] (Match reserved areas touching specified rectangle) [-filter expression] (Filter collection with 'expression') [-quiet] (Suppress all messages) patterns (Match route guides)</pre>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
none	<pre>get_route_shapes # Creates a collection of route shapes. [-within rectangle] (Match reserved areas within specified rectangle) [-region region] (Match route shapes intersect specified rectangle) [-at at_point] (Match route shapes overlap the specified point) [-filter expression] (Filter collection with 'expression') -of_objects net_list (Get route shapes of these nets) [-quiet] (Suppress all messages) patterns (Match route shapes)</pre>
none	<pre>get_scheme_var # Gets the value of a Scheme variable. scheme variable name (Variable to be examined)</pre>
none	<pre>get_selection # Gets objects that were selected graphically.</pre>
dbGetPinByName cellId "pinName"	<pre>get_terminals # Creates a collection of terminals. [-of_objects port_list] (Gets the terminals of the ports) -object_id integer (Gets the port shape using its Milkyway object ID; defaults to work in current_design) [-filter expression] (Filter collection with 'expression') [-quiet] (Suppresses all messages) [-regexp] (Patterns are full regular expressions) [-exact] (Wildcards are considered as plain characters) [-nocase] (With -regexp, matches are case-insensitive) patterns (Matches terminal names against patterns)</pre>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
none	<pre>get_texts # Creates a collection of text. [-quiet] (Suppresses all messages) [-within rectangle] (Specifies that the bounding box of the text must be within the specified rectangle) [-touching rectangle] (Specifies that the bounding box of the text must touch the specified rectangle) [-intersect rectangle] (Specifies that the bounding box of the text must intersect with the specified rectangle) [-at point] (Specifies that the bounding box of the text must overlap with the specified rectangle) [-filter expression] (Specifies the filter collection with 'expression') [patterns] (Matches text names against patterns. Default: *)</pre>
none	<pre>get_up_objects # Creates a collection of uninterpreted physical objects. [-within rectangle] (Matches reserved areas within a specified rectangle) [-region region] (Matches the net shapes intersect with the specified rectangle) [-at at_point] (Matches the net shapes overlap with the specified point) [-filter expression] (filter collection with 'expression') [-quiet] (Suppresses all messages) [patterns] (Matches uninterpreted objects)</pre>
dbFetchContactNames	<pre>get_via_masters # Gets via masters in the current library. patterns (Matches via master names against patterns)</pre>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
none	<pre>get_voltage_areas # Creates a collection of voltage areas. [-quiet] (Suppresses all messages) [-within rectangle] (Specifies that the bounding box of the voltage area must be within the specified rectangle) [-touching rectangle] (Specifies that the bounding box of the voltage area must touch the specified rectangle) [-intersect rectangle] (Specifies that the bounding box of the voltage area must intersect with the specified rectangle) [-at point] (Specifies that the bounding box of the voltage area must overlap with the specified rectangle) [-filter expression] (Filter collection with 'expression') [-of_objects cell_list] (Gets the voltage areas of these cells) [patterns] (Matches the voltage area names against patterns. Default is *)</pre>
none	<pre>index_collection # Extracts an object from a collection. The result is a new collection. collection1 (Collection to index) index (Index (zero based): Value >= 0)</pre>
dbFetchLayerTypes	<pre>list_attributes # Lists currently defined attributes.</pre>
dbFetchObjectField fileId objectId fieldName ->? id	<pre>[-application] (List application attributes as well) [-class class_name] (List attributes of this class only: Values: port, cell, pin, net, design, lib, lib_cell, lib_pin, terminal, route_guide, placement_blockage, net_shape, route_shape, bound, clock)</pre>
dbFetchTechTypes	

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
none	mem # Specifies the memory used.
dbSetCellInstOrigin cellId “cellInstName” origin	move_objects # Moves one or more objects.
?dbMoveCellOrigin cellId origin	-delta vector (Specifies the displacement for objects to move through) -to point (Specifies the new location of the reference point) [-from point] (Specifies the reference point on the objects)
?dbSetCellOrigin cellId “justification”	[-redraw] (redraw after movement) objects (Specifies the objects to move)
dbOpenCell “cellName” “mode”	open_design # Opens the design. [-library library] (Library from which the design will be opened) [-readonly] (Open design for read only) design_name (Design to be opened)
dbOpenLibrary “libName” “mode”	open_mw_lib # Opens a Milkyway library. [-readonly] (Open library for read only) [-write_ref] (Open reference library for write) mw_lib (Milkyway library to open)

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
none	<pre>purge_attached_file # Purges the attached files in the library. [-effort { low medium high }] (Specifies the effort to purge the attached files) [-files {<pseudoname> <design1> <design2> ... }] (Specifies the list of attached files to purge) [-force] (Specifies the force to link the specified attached files) [-reset_index] (Resets the attached files' index) library_name (Specifies the library in which to purge attached files)</pre>
none	<pre>query_objects # Displays objects in the database. [-verbose] (Display type of each object) [-truncate elem_count] (Truncate display to n elements) [-class class_name] (If object_spec is a name, this is its class) obj_spec_list (Objects to find and display)</pre>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
read_def	<pre>read_def # Reads the DEF file into the Milkyway database. -lib library_name (Specifies the library name) -design design_name (Specifies the design name) [-enforce_scaling] (Specifies the enforce scaling) [-advance_cell_version] (Keeps the current cell version) [-netl_phys incr_incr incr_rimport rimport_rimport nochange_incr nochange_rimport] (netl_phys; Default: incr_incr) [-site site_def_file_name] (Specifies the site definition file name) [-core core_site_name] (Specifies the core site name) [-h_layer h_layer_list] (Specifies the list of horizontal layers; available horizontal layer: m1, m2, m3, m4) [-turn_via_to_inst] (Translates turn vias to paths (default) or cell insts) [-inexactly_matched_via_to_inst] (Translates inexactly matched vias to contact arrays and wires (default), or cell insts) [-lef lef_file_name] (Specifies the LEF file name) def_file_name (Specifies the DEF file name)</pre>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
read_pdef	<pre>read_pdef # Reads a PDEF file into the Milkyway database. -design design_name (Specifies the design name) [-pin_mode overwrite incremental] (Specifies the pin mode; Default: overwrite) [-skip_signal_nets] (Skips signal nets) [-inexactly_matched_via_to_inst] (Translates inexactly matched vias to contact arrays and wires (defalut), or cell insts) pdef_file_name (Specifies the PDEF file name)</pre>
none	<pre>remove_attribute # Removes attribute on the specified object. [-class string] (Class name for the patterns) [-quiet] (Do not report any messages) object_list (Objects on which to remove the attribute) attribute_name (Name of the attribute)</pre>
none	<pre>remove_bounds # Removes a list of bounds. -all (Specifies to remove all bounds) -name list (Specifies the name of the bound) bound_list (Specifies the list of move bounds to remove)</pre>
dbPurgeCellInstByName cellId “cellInstName”	<pre>remove_cell # Removes cells.</pre>
dbPurgeCellInstMaster cellId “cellInstMasterName”	<pre>-master master_list (Master list from which cells are to be removed) -all (All cells will be removed) [-verbose] (Be verbose) cell_list (Cell list to be removed)</pre>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
none	remove_collections # Removes all collections.
dbDeleteCell “cellName”	remove_design # Removes designs from the current library. [-version_kept count] (Specifies how many recent versions to keep) [-verbose] (Be verbose) [-all_versions] (Specifies to remove all versions of the specified design) design_list (List of designs to be removed)
none	remove_from_collection # Removes objects from a collection. The result is a new collection. collection (Base collection) object_spec (Objects to remove)
dbPurgeNetByName cellId “netName”	remove_net # Removes nets in design. [-design design name] (Remove nets in specified design) [-all] (Remove all nets in design) list of net (Names of nets to remove)
dbPurgePin cellId pinId	remove_net_shapes # Removes wire net shapes net shapes. [-verbose] (Be verbose) net_shapes (Collection of handles of wire net shapes to be removed)

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
dbPurgeObject cellId objectId	remove_objects # Removes objects from design.
dbDeleteObject cellId objectId	-class class (Class of which to remove objects) object_id (Object ID to be removed)
dbPurgeAllPlanGroup cellId	
dbDeleteOneLayer Rectangle “libName” “layerNumber”	remove_placement_blockage # Removes a collection of placement blockages. -all (Option to remove all placement blockage) [-verbose] (Be verbose) patterns (Collection of placement blockage to be removed)
dbPurgePort cellId portId topCellName	remove_port # Removes ports. [-verbose] (Be verbose) ports (List of ports to be removed)
dbReplaceRefLibByName libId “oldRefLibName” “NewRefLibName”	remove_reference_library # Removes reference libraries. [-from mw_lib] (Main library to remove from) [-lib_id lib_id] (ID of the main library: Value >= 1)
dbDeleteRefLibByName libId “refLibName”	-all (Remove all reference libraries) libraries (List of reference libraries to remove)

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
none	<pre>remove_route_guide # Removes a collection of route guide. -all (Option to remove all route guides) [-verbose] (Be verbose) patterns (Collection of route guides to be removed)</pre>
none	<pre>remove_route_shape # Removes route shape. [-verbose] (Be verbose) route_shapes (Collection of route shapes to be removed)</pre>
none	<pre>remove_symbol_table # Frees up memory from Tcl symbol tables.</pre>
dbPurgePin cellId pinId	<pre>remove_terminal # Removes a collection of terminals. [-verbose] (Be verbose) terminals (Collection of terminals to be removed)</pre>
none	<pre>remove_text # Removes a list of texts. -all (Specifies to remove all texts) text_list (Specifies the list of texts to remove)</pre>
none	<pre>remove_voltage_area # Removes a list of voltage areas. -all (Specifies to remove all voltage areas) voltage_area_list (Specifies the list of voltage areas to remove)</pre>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
dbRenameCell oldCellName newCellName	rename_design # Renames the design to a new name. old_name (Specifies the design to be renamed) new_name (Specifies the new name of the design) [-all_version] (Indicates to rename all versions)
dbRenameLib “oldLibName” “newLibNAme”	rename_mw_lib # Renames a Milkyway library. -from lib_name (Name of the library to be renamed) -to lib_name (New name of the library)
dbReplaceCellInst cellId instId “newCellName” “newViewName” “topCellName”	replace_cells # Replaces cell in the current design. -design new_design (New design to be used) [-verbose] (Be verbose) cell_list (Cell list to be replaced)
none	replace_tlu_plus_file # Replaces the TLUPlus file for the library. [-library mw_libs] (Specifies the list of libraries to set the TLUPlus file) [-lib_id lib_id] (Specifies the ID of the library to set the TLUPlus file: Value >= 1) tlu_plus_file (Specifies the TLUPlus file to be set)

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
none	<pre>report_attribute # Reports the attributes on one or more objects. [-class one-of-string] (Specifies the class name for the patterns) [-application] (Specifies the list application attributes as well as the user-defined attributes) [-quiet] (Specifies not to report any messages) object_list (Specifies the objects that are to report their attributes)</pre>
none	<pre>report_collections # Reports data about Tcl collections.</pre>
dbGetAllLibCellInfo “libName”	<pre>report_design # Report information of designs.</pre>
dbGetOneLibCellInfo “libName” “cellName”	<pre>[-quiet] (Suppress all messages) [design_list] (Designs to be reported)</pre>
dbDumpCell cellId “fileName”	
dbDumpCellPlacement cellId “cellType” “fileName”	

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
dbListUnitRange “libName”	report_mw_lib # Reports various kinds of information about a Milkyway library.
dbDumpTechFile libId fileName	[-unit_range] (Report unit range of library) [-technology] (Report technology of library) [-output file_name] (Technology file name to report to) [-lib_id lib_id] (ID of the library to be reported: Value >= 1) [mw_lib] (The Milkyway library to be reported)
none	rotate_objects # Rotates one or more objects. -to orientation (Specifies the new orientation) -by orientation (Specifies the change in orientation) [-pivot point] (Specifies the point for pivoting) [-fixed_ll] (Fixes the lower-left corner of the object during rotation) [-redraw] (Redraws after movement) objects (Specifies the objects to rotate)
dbSaveCell cellId	save_design # Saves the design. [-verbose] (Be verbose) [design_list] (Designs to be saved)

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
none	<pre>scheme # Runs a single Scheme command. scheme_command_name (Command to be executed) [strings ...] (Arguments for Scheme command, if any) When running in Tcl mode, you have the option of interpreting a Scheme function by prefixing it with the "scheme" command. The Scheme function must be a single element, which implies that you must generally enclose the command in braces {}. Use the "begin_scheme" command for longer sequences of Scheme commands.</pre>
none	<pre>set_attached_file_link_mode # Sets or checks the attached file's link mode. [mode] (Specifies the value to set)</pre>
dbCreateCellBoundary cellId pointArray	<pre>set_attribute # Sets attribute data on the specified object.</pre>

dbSetCellPortTypes
libName cellName
portList ?append

dbSetStdCell cellId

dbSetIOPad cellId

dbUpdateNetStatus
cellId

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
none	<pre>set_hierarchy_separator # Specifies the character to use for hierarchy seperation. hier_char (Character to use)</pre>
none	<pre>set_mw_lib_bus_naming_style # Sets the bus naming style for the library. [-library mw_libs] (Specifies the libraries for which to set the bus naming style) [-lib_id lib_id] (Specifies the ID of the library to set the bus naming style: Value >= 1) string (Specifies the bus naming style to set on the library)</pre>
none	<pre>sizeof_collection # Gets the number of objects in the collection. collection1 (Elements in this collection)</pre>
none	<pre>sort_collection # Creates a sorted copy of the collection. [-descending] (Sort in descending order) collection1 (Collection to sort) criteria (Sort criteria - list of attributes)</pre>
none	<pre>start_gui If you start Astro in the -no_gui mode, you will operate from the original text window. To bring up a GUI window, use the “start_gui” operation. See Also: stop_gui</pre>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
none	<p>stop_gui</p> <p>If you want to terminate GUI operation and return control to the window that you used to invoke Astro, use the “stop_gui” command.</p> <p>See Also: start_gui</p>
none	<p>undefine_user_attribute</p> <p>#Undefines a user attribute.</p> <p>-class (Specifies the class type for the object on which to undefine the attribute)</p> <p>-quite (Specifies not to report any messages)</p> <p>attr_name (Specifies the name of the attribute)</p>
dbAddCellInstToCell Region cellId regionId cellInstId dbDeleteCellInstFromCell Region cellId regionId cellInstId dbAddCellInstToPlan GroupName cellId pgId “ciName”	<p>update_bound</p> <p># Modifies information for an existing bound.</p> <p>-add list (Specifies the cells to be added to the bound)</p> <p>-remove list (Specifies the cells to be removed from the bound)</p> <p>bound (Specifies a bound to be updated)</p>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
dbUpdateHierarchy cellId	update_design # Updates the specified design to make it consistent.
dbForceUpdateHierarchy cellId	[-in_dir dir_name] (Updates all designs of all the libraries in the specified directory)
dbRecursiveUpdateAll Hierarchy cellId	[-hierarchy] (Hierarchy of the designs to be updated) [-recursive] (Update all referenced designs)
dbForceRecursiveUpdate AllHierarchy cellId	[-force] (Check if there are extra ports when updating) [-bbox] (Updates all database page bounding boxes of the specified design)
dbForceUpdateHierarchy InDir dirName	[-verbose] (Be verbose) [design] (Designs to be updated)
dbForceRecompCell BBox cellId	
dbReplaceTechFile libId “techFile”	update_mw_lib # Updates a Milkyway library.
dbSetTechFileName “techFileName”	[-rebuild] (Rebuilds the library) [-technology file_name] (Updates the technology information)
dbRebuildLib “libName”	[-lib_id lib_id] (Specifies the ID of the library to be updated: Value >= 1) [mw_lib] (Specifies the name of the library to be updated)

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
write_def	<pre>write_def # Writes a DEF file from the Milkyway database. -lib library_name (Specifies the library name) -topdesign design_name (Specifies the design name) -output output_file_name (Specifies the output file name) [-version 5.3 5.4 5.5] (Specifies the DEF version; default: 5.5) [-unit conversion_factor] (Specifies the conversion factor) [-compressed] (Specifies compressed) [-rows_tracks_gcells] (Specifies the output rows, tracks, and gcells) [-vias] (Specifies the output vias) [-all_vias] (Specifies to output all design vias) [-lef lef_file_name] (Specifies the LEF file name; default : MW.lef.inc) [-regions_groups] (Specifies the output regions and groups) [-components] (Specifies the output components) [-macro] (Specifies the output macro cells) [-fixed] (Specifies the output fixed cells) [-placed] (Specifies the output placed cells) [-pins] (Specifies the output pins) [-blockages] (Specifies the output blockages) [-specialnets] (Specifies the output special nets) [-nets] (Specifies the output nets) [-routed_nets] (Specifies the output routed nets) [-diode_pins] (Specifies the output diode pins) [-scanchain] (Specifies the output scanchains) [-notch_gap] (Specifies the output notch gaps) [-floating_metal_fill] (Specifies the output floating metal fills) [-pg_metal_fill] (Specifies the output P/G meal fills)</pre>

Table 1-2 Mapping Scheme to Synopsys Tcl Commands (Continued)

Scheme commands	Synopsys Tcl commands and syntax
write_pdef	<pre>write_pdef # Writes a PDEF file from the Milkyway database. -design design_name (Specifies the design name) -output output_file_name (Specifies the output file name) [-unit distance_unit] (Specifies the distance unit) [-site_map_file site_mapping_file] (Specifies the site mapping file) [-fixed_only_cell] (Specifies to only ouput the fixed cell) [-std_cell] (Specifies the output std cell) [-macro_cell] (Specifies the output macro cell) [-io_pad_cell] (Specifies the output io pad cell) [-coner_pad_cell] (Specifies the output coner pad cell) [-std_filler_cell] (Specifies the output std filler cell) [-pad_filler_cell] (Specifies the output pad filler cell) [-skip_signal_nets] (Skips signal nets)</pre>

Scheme Commands Not Supported in Tcl

The following commands are not available as Tcl-equivalent commands.

```
clfCreateTable
clfCreateVectorTable
clfDefineDesignCellName
clfGetCLFLibId
clfGetCLFLibName
clfPurgeBooleanFunction
clfPurgePad
cmdGetLastError
cmdGetLastErrorMsg
cnvGetBuffering
cnvSetBuffering
cPin
cPinF
dbDefineCTSPort
dbDefineIgnoreP*
dbDefineSyncP*
dbSetInputDelay
dbSetOutputDelay
define3dCondTimeTLU
define3dTImeT*
defineAntennaArea
defineBooleanFunction
defineCellDontTouch
defineCellDontUse
defineCellFanoutLoad
defineCellFootPrint
defineCellMaxCapacitance
defineCellMaxFanout
defineCellMaxTransition
defineCellMinCapacitance
defineCellMinFanout
defineCellMinTransition
defineClockNetworkTLU
defineCondMinClkPeriod
defineCondMinClkPulseWidth
defineCondTimeIntrinsic
defineCondTimeRemovalIntrinsic
```

```
defineCondTimeRemovalTLU
defineCondTimeSkewIntrinsic
defineCondTimeSkewTLU
defineCondTimeTLU
defineCurrentWaveform
defineDiodeProtection
defineEEQCell
defineFlipFlopFunction
defineFootPrintEQCell
defineGateSize
defineGeneratedClock
defineHierAntennaProp
defineInternalPower
defineLatchFunction
defineLeakagePower
defineLEQCell
defineLibAttributes
defineMinClkPeriod
defineMinClkPulseWidth
definePad
definePortCapacitance
definePortDriveTLU
definePortFeedthru
definePortPhaseDelay
definePortTieHigh
definePortTieLow
definePowerValue
defineStaticNoiseMargin
defineStaticNoiseWidth
defineSupplySignalRelation
defineSwitchingPower
defineTableTemplate
defineTimeIntrinsic
defineTimeTLU
defineTristateFunction
fcPad
flipChipMatrix
flipChipPad
getTclVar
hardFence
insertPad
loadUserForm
loadUserFormFromPath
markNet
netCapacitance
```

```
netDelay
netLength
netTransition
netWeight
RPCCommands
RPCConnect
RPCDump
RPCEstPrimaryConnect
RPCGetSession
RPCKill
RPCLogCommand
RPCPing
RPCProcessExists
RPCQueryConnections
RPCRemote
RPCSendMsg
RPCSetServerMode
RPCSetSession
RPCTestMode
RPCUnremote
rPin
scmxConcatFunction
scmxPreprocess
scmxSetLogMode
scmxThrow
sendCommand
setNetVariableWidth
softFence
tclLoad
tdfPurgeDrivingCell
tdfPurgeFlipChipMatrix
tdfPurgeInputDelay
tdfResetPath
tdfSetClkDef
tdfSetClockGatingCheck
tdfSetClockPhaseDelay
tdfSetClockPortTransition
tdfSetClockWaveform
tdfSetDerivedClk
tdfSetDisableTiming
tdfSetDrivingCell
tdfSetFalsePath
tdfSetGeneratedClk
tdfSetInputDelay
tdfSetInputPortTransition
```

```
tdfSetLoadCap
tdfSetLoadCapacitanceList
tdfSetMaxDelayPath
tdfSetMinDelayPath
tdfSetMinIOSpacing
tdfSetMinPulseWidth
tdfSetMultiCyclePath
tdfSetNetCapAndTransTime
tdfSetNetCapTransAndDelayTime
tdfSetNetTransitionDelay
tdfSetNoTouchCellInst
tdfSetNoTouchNet
tdfSetOutputDelay
tdfSetPowerSupply
tdfSetRestoreTiming
tdfSetSrcCapacitance
tdfSetSrcCapacitanceList
tdfSetSrcResistanceList
tdfSetTimeBorrowLimit
userFormCancel
userGetTclVar
varValue
```


2

Scheme Fundamentals

The Scheme Programming Language, or Scheme, is a high-level, easy to use, programming language that supports operations on the following types of data:

- Strings
- Lists or vectors of objects
- Vectors
- Numbers
- Characters

Scheme provides an extensive set of data types and flexible control structures, which makes it very versatile. Because machine dependencies are virtually hidden from the developer, Scheme programs are very portable across implementations of the same Scheme system on two different platforms.

This chapter explains the fundamentals of Scheme. For additional information on the language, see the *Report on the Algorithmic Language Scheme*.

Lexical Conventions

This section explains the lexical conventions used in Synopsys Scheme.

Identifiers

There are no true keywords in Synopsys Scheme. Even fundamental Scheme constructs are implemented as intrinsic functions. Further, intrinsic functions within Synopsys applications are predefined and unchangeable in the top-level environment. For example,

```
(define let 1)
```

results in an error because *let* is an intrinsic function. However, as described in “[Internal Definitions](#)” on page 2-20, Scheme permits intrinsic functions within Synopsys applications to be used as variables. Synopsys product specialists strongly discourage using intrinsic functions as variables, and if this capability is removed in a future release, it is likely there will be no backward compatibility.

The characters ? and ! have no special properties. By convention, predicate procedures (procedures that return boolean values) are named by identifiers that end in ? and most data mutation procedures are named by identifiers that end in !.

Unlike most Scheme implementations, Synopsys Scheme identifiers are case sensitive. Further, Synopsys Scheme allows identifiers of the form +x. Most Scheme implementations do not allow the symbols + and - to be the first character of a symbol.

Numbers

Synopsys applications recognize the following Scheme notations for numbers:

- The number with no radix specified (base 10 is assumed)
- `#b<number>` to express a number in base 2
- `#o<number>` to express a number in base 8
- `#d<number>` to express a number in base 10
- `#x<number>` to express a number in base 16

Examples of valid numerical constants follow:

```
#d1.1e1
 11
1.1e2
 110
11
 11 ; integer
#xf
 15
#b10
 2
```

Only decimal numbers support non-integral values and exponentiation. If you incorrectly use non-integral values and exponentiation, unexpected results may occur. For example,

```
#b1.01
 #b1 .01
#o7e5
 7 e5
```

Note:

The lexicographical analyzer does not generate an error for these constructs.

Characters

Synopsys applications recognize the following Scheme notations for characters:

#*character*

For example,

```
#\a;lower case letter  
#\A;upper case letter  
#\(;left parenthesis
```

Note:

The form #\<characterName> is not recognized

White Space and Comments

Synopsys Scheme supports all standard conventions applying to white space. White space characters are spaces, tabs, newlines, and formfeeds. White space can occur between variables or numbers, and inside strings for improved readability. However, white space cannot occur within an identifier.

Synopsys Scheme supports the following nonstandard delimiters for white space within a string constant:

- \n specifies a newline.

- `\f` specifies a formfeed.
- `\t` specifies a tab.

Synopsys Scheme supports all standard conventions applying to comments. To add comments, place a semicolon (`;`) where you would like the comment to begin. The comment then continues to the end of the line on which the semicolon appears. In Scheme, the comment acts as white space.

Other Notations

The following Synopsys notations have the same meaning as standard Scheme:

- Parentheses `()` are used for grouping and notating lists.
- An apostrophe `(')` is used to quote literal data.
- Quotation marks `("")` are used to delimit strings.
- The characters `#t` and `#f` are used to represent the boolean constants true and false, respectively.
- The backslash `(\)` is used as an escape character within string constants.
- Brackets `([])` are reserved for possible future extensions to the language and are not currently supported.

The following are exceptions to standard Scheme:

- The outermost parentheses `()` are not required around a statement when the entire statement is on a single line. For example, the following are equivalent:

```
(+ 1 2)
+ 1 2
```

- The plus sign (+) and minus sign (-) are allowed as the first character of an identifier.
- The back quotation mark (') is not supported.
- The comma (,) and the sequence comma-at sign (@) are not supported.
- Vector constants are not supported.
- Certain object types can be terminated by nonwhite space. For example,

```
define x (1abc)
ž(1 abc)
```

Note:

Later releases might not support this convention. Delimit all tokens by proper delimiters such as blanks, tabs, newlines, left parenthesis ((), right parenthesis ()) , and quotation marks (" ") to prevent any future problems.

Basic Concepts

This section explains the Synopsys implementation of standard Scheme basic concepts.

Variables

Synopsys Scheme supports the standard Scheme definition for variables. Variables, which can be any acceptable identifier (see “[Identifiers](#)” on page 2-3), name a location that stores a specified

value. A variable is bound to that location. Lambda expressions (see “[Lambda Expressions](#)” on page 2-14) and other binding constructs, such as *let*, *let**, and *letrec* expressions (see “[Binding Constructs](#)” on page 2-17), create new locations and bind variables to those locations.

Variables have a static scope inside Scheme. Bound variables have a region in which the binding applies. This region is determined by the expression, such as *lambda*, or other binding construct that is used to bind the variable. When a variable is not bound in the region in which it is used, Synopsys applications refer to the top-level definition of the variable (see “[Top-Level Definitions](#)” on page 2-19). If there is no top-level definition for the variable, Synopsys applications signal an error.

It is possible for a symbol to have multiple values within different regions. For example, the symbol *x* could have the value 5 as a top-level definition and have the value “blue” inside of a *let* construct. Inside the body of the *let* construct the value of *x* is “blue.” Outside the body of the *let* construct the value of *x* is 5. Variables defined by these binding constructs do not persist beyond the body of the binding construct. A slight difference to standard Scheme is that a variable defined with a *define* function inside the scope of a binding construct still creates a top-level definition, or if the variable exists, *define* modifies the value of the variable within the scope rules.

The Evaluator

The evaluator processes the next list object supplied by the parser. The parser adds a balancing set of parentheses to make the line a valid list object. For example, suppose the following appears on a line by itself:

```
define x (+ 1 2)
```

The evaluator then gets the list

```
(define x (+ 1 2))
```

Parentheses are necessary because the evaluator evaluates a symbol as a function only if it is to the right of a left parenthesis (in other words, if it is the first element of the list). If you want to conform to standard Scheme syntax exactly, always use the second form.

Once the evaluator has the list, it recursively evaluates the list object, starting at the leftmost object. In the preceding example, the symbol *define* is the leftmost object. The symbol *define* is bound to an internal function, which requires two arguments. The evaluator checks the length of the list to ensure that it contains exactly two more objects, and it does, object 1, x, and object 2, (+ 1 2).

Next the arguments are generally evaluated. In the case of internal functions, you can set a flag to mark the argument in a special way, as “don’t evaluate.” In the case of *define*, you do not want to evaluate the value of the symbol x. The purpose of *define* is to make that relationship. So, argument 1 needs no further processing. It is passed to *define* as just the symbol x.

The second argument, (+ 1 2), does not have this special flag set. It is evaluated. The same process begins. The argument (+ 1 2) is a list. Its leftmost object is the symbol +, which again is an internal function. The function + takes two arguments, both numbers. The first object in the list is the numeric integer constant 1. It is evaluated, and the result of evaluating a constant is that constant. The second argument is evaluated, and again, the result of evaluating the integer constant is that constant. All of the arguments to the function + have been evaluated.

Next the evaluator checks the types of both arguments to make sure they match the `+` template, and they do. The internal function `+` is called with the integer atoms `1` and `2`. The result of the `+` function is the integer constant `3`. Now that all of the arguments to `define` that need evaluation have been evaluated, its arguments are type checked. The first object must be a symbol, and it is, and the second argument can be anything. The internal function `define` is called. Because `define` was called from the top level, the value `3` (the result of the `define` function) is printed.

For user-defined functions, the process is very similar. The only difference is that the evaluator does not perform a type check of the arguments.

In the case of some variable argument functions, the evaluator does not perform a check on argument count. In these cases, it is the task of the actual internal C function or user-defined function to verify that the argument count is correct.

Errors

Upon discovery of an error, the evaluator stops evaluation and prints both an error message and the current stack of functions. For the following example, assume that the variables `cell1` and `net1` are not bound.

```
define lib1 (dbOpenLib "train")
dbOpenCell "toprt" "w"
.
.
.
dbCreatePort cell1 net1 "power"
ERROR: variable cell1 not bound.
Stack follows
dbCreatePort.
()
```

In this example,

- The first line opens a library for editing.
- The second line opens a cell for editing.
- The third line attempts to create a port. However, the Synopsys application discovers that the variable *cell1* is not bound, stops evaluation, and displays an error message.

The error message informs you that the variable *cell1* is not bound and that the error occurred during execution of *dbCreatePort* (see the *IC Implementation Command Reference Manual*).

Note:

Evaluation stops upon discovery of the first error. Therefore you would not be informed if the variable *net1* were not bound.

For more information about the commands presented in this example, see *dbOpenLib*, *dbOpenCell*, and *dbCreatePort* in the *IC Implementation Command Reference Manual*.

It is possible to force an error in Scheme with the function *signal-interrupt-error*. You use this function when you want to cancel program execution.

Interrupts

The evaluator checks to see whether a user-signalled interrupt (generally mapped to the ^C key) has occurred every *n* seconds of CPU time. The value of *n* can be set with the Scheme function *setInterruptPollInterval*. Because of certain other administrative tasks performed by the evaluator, this check might be performed

more frequently. The actual time to respond to the interrupt varies depending on what the program is executing at that moment. If the program is evaluating Scheme, the response is almost immediate (the next object evaluated). If the program is performing an algorithm, such as placement, the response to the interrupt can be much longer. The actual response to the interrupt depends upon the setting of the interrupt mask. Scheme allows the setting of the mask to

- Mask all interrupts with respect to non-Scheme parts of the system. This mask would mask interrupts to the placement kernel, for example. Use

```
(setInterruptMask (CONSTANT "scbSysMaskInterrupt"))
```

to set this mask. Use care when setting this mask. Placement and other place and route system operations cannot be interrupted when this mask is set.

- Mask all interrupts from the evaluator. Replay is still interruptible and would be interrupted at the end of a replay step. Certain loads, such as constraints, are also interrupted. Use

```
(setInterruptMask (CONSTANT "scbEvalMaskInterrupt"))
```

to set this mask.

- Mask all interrupts by using

```
(setInterruptMask (bit-or (CONSTANT  
"scbEvalMaskInterrupt")  
(CONSTANT "scbSysMaskInterrupt"))))
```

Normal interrupts (when unmasked by *scbEvalMaskInterrupt*) cause the evaluator to force an exception and terminate execution. The Scheme function *interrupt?* queries whether there has been an interrupt. This function always returns the state of interrupt,

regardless of any masks in place. The Scheme function *signal-interrupt-error* actually generates an error to Scheme, not an interrupt. Because this function signals an error, the state of the interrupt masks is ignored. By masking interrupts to the evaluator and using the *interrupt?* function, you can program the processing of interrupts.

True and False

A Synopsys application does not interpret the empty list as equal to false. For example,

```
equal? '() #f
```

is false. However, the '()' value is treated as false in Boolean expressions.

Expressions

This section explains the Synopsys implementation of Scheme expressions. A Scheme expression is a construct, such as a variable reference, literal expression, procedural call, or conditional expression, that returns a value.

Primitive Expression Types

This section explains the Synopsys implementation of primitive expression types.

Variable References

Synopsys Scheme supports the following essential syntax for variable references:

variable

Caution!

It is an error to reference an unbound variable.

Literal Expressions

Synopsys Scheme supports the following essential syntax for literal expressions:

```
quote datum
' datum
literal_constant
```

Both *quote datum* and *'datum* evaluate to *datum*. The *datum* argument can be any symbol.

Note:

The literal expressions *quote* and “ ‘ ” are equivalent.

Procedure Calls

Synopsys Scheme supports procedure calls with the syntax

```
(operator {operand} ... )
```

Lambda Expressions

Synopsys Scheme supports the following syntax for lambda expressions:

lambdaformals body

where the arguments are as follows.

body

A sequence of one or more expressions.

formals

Either of the following:

```
variable  
(variable1 variable2 ...variablen)
```

The first method implies that the function definition accepts a variable number of arguments. When the function is called, all of the operands are packaged in a list, and the variable is bound to that list.

The second method implies that the function accepts a fixed number of arguments. The variable *variable₁* is bound to *operand₁*, the variable *variable₂* is bound to *operand₂*, and so on. If the number of operands do not match the number of variables, an error occurs.

body

A sequence of one or more expressions.

An exception to standard Scheme is that the Synopsys implementation of Scheme does not support the dotted syntax for *formals*:

```
(variable1 ... variablen-1 . variablen)
```

Conditionals

Synopsys Scheme supports the following syntax for conditional expressions:

```
if test consequent alternate  
if test consequent
```

The arguments *test*, *consequent*, and *alternate* can be any single arbitrary expression. If the result of *test* is not *false* and not *nil*, *consequent* is evaluated and returned. If the result of *test* is *false* or *nil*, *alternate* is evaluated and returned. Only one of *consequent* or *alternate* is evaluated.

Assignments

Synopsys Scheme supports the following essential syntax for assignment expressions:

```
set! variable init
```

Note:

The function *set!* does not create variable. Unless a previous step binds variable, *set!* results in an error. For example,

```
set! x 4  
ERROR: variable x not bound.
```

Derived Expression Types

This section explains the Synopsys implementation of derived expression types.

Conditionals

Synopsys Scheme supports the following essential syntax for derived expression conditionals:

```
cond { (test expression ...) } ...  
cond { (test expression) } ... (else expression ...)
```

The tests are evaluated from left to right. The first *test*, which is not *nil* and not *#f*, is taken, and all of the expressions within that test's sublist are evaluated. The last expression evaluated is returned. If no tests are taken, the *else* clause is evaluated. If no *else* clause exists, the result of *cond* is *nil*, and no expressions are evaluated (only the test clauses are evaluated).

Binding Constructs

Synopsys Scheme supports the following binding constructs:

```
let ( {variable init} ... ) body  
let* ( {variable init} ... ) body  
letrec ( {variable init} ... ) body
```

where the arguments are as follow:

variable

Identifier to which *init* is bound.

See “[Identifiers](#)” on page [2-3](#) for restrictions applying to variable names.

init

Value or expression to which *variable* is initially set.

body

A sequence of one or more expressions.

Although the binding constructs *let*, *let**, and *letrec* have the same syntax, their bindings are effective in different regions.

- The *let* function evaluates the initial values (*init1*, *init2*, ..., *initn*) in the current environment and binds the variables to new locations. There can be no interdependency between the initial

values ($init_1, init_2, \dots, init_n$). Then *let* evaluates *body* in the extended environment and returns the last expression of *body*. The *body* argument is the region in which the bindings are effective.

- The *let** function is shorthand for nested *let* functions. This function performs bindings sequentially from left to right. The *let** function performs each successive binding in a region, *body*, where the previous bindings are visible. For example, a second binding is done in an environment where the first binding is visible. The *body* argument is the region in which the bindings are effective.
- The *letrec* function binds the variables to new locations and fills them with undefined values. Then *letrec* evaluates the variables to the specified initial values. The entire *letrec* expression is the effective region for all bindings. This makes it possible to define mutually recursive procedures.

Note:

When *letrec* is used, all initial values ($init_1, init_2, \dots, init_n$) must evaluate without reference to the value of any of the other variables.

Sequencing

Synopsys Scheme supports the following essential syntax for sequencing expressions:

```
begin {expression} ...
```

The *begin* expression essentially makes a single expression out of multiple expressions. It evaluates the expressions in order, from left to right, and returns the value of the last expression. For example,

```
begin (display "4 plus 1 equals") (display (+ 4 1))  
      "4 plus 1 equals"5()
```

Iteration

Synopsys Scheme supports the following syntax for iteration expressions:

```
do ({(variable init step)} ...) (test {expression} ...)
{command} ...
```

The *do* construct is used to sequence through a set of values. The *variable* variable is initialized to the evaluated value of *init*. Next *test* is evaluated. If the result is *#f* or *nil*, *command* is evaluated. The variables are assigned the result of *step*, and *test* is evaluated again. This loop iterates until the evaluation of *test* results in not *#f* or *nil*. Then expression is evaluated. The result of *do* is the last expression.

Delayed Evaluation

Synopsys Scheme does not support delayed evaluation expressions.

Quasiquotation

Synopsys Scheme does not support quasiquotation expressions.

Definitions

This section explains the Synopsys implementation of standard Scheme definitions.

Top-Level Definitions

Synopsys Scheme supports the following syntax for top-level definitions:

```
define variable init
```

Unlike the `set!` function (see “[Assignments](#)” on page 2-16), if `variable` has not been bound previously to a location, `define` creates `variable`.

Caution!

The Synopsys Scheme `define` is slightly different from the standard Scheme `define`. In standard Scheme, `define` binds a new variable in the current region if the named variable does not exist. In Synopsys Scheme, `define` defines a top-level variable if the variable does not exist. If the variable exists in the current environment, `define` behaves the same as in standard Scheme. The `define` function has the same effect as a `set!`. If your intent is to create a variable within the current region, use `let` (see “[Binding Constructs](#)” on page 2-17) instead of `define`.

Internal Definitions

Synopsys Scheme supports internal definitions. Internal definitions occur at the beginning of the `body` field for `let`, `let*`, and `letrec` constructs (see “[Binding Constructs](#)” on page 2-17). Variables defined in this manner are effective only in the region specified by the given function. For example,

```
(  
 (let*  
 (  
 (cell1 (dbOpenCell "toprt" "r" ))  
 (net1 (dbGetNetByName cell1 "VDD" ))  
 )  
 .  
 .  
 .  
 )  
 dbCreatePort cell1 net1 "power"  
 )  
 ŸERROR: variable cell1 not bound.  
 Stack follows  
 dbCreatePort.  
 ()
```

In this example, the variables *cell1* and *net1* are created and assigned values inside the *let** statement. After a Synopsys application evaluates the *let** statement, these variables no longer exist.

Boolean Procedures

This section explains the Synopsys implementation of standard Scheme built-in procedures. Unless specifically stated to the contrary, all functions require a fixed number of arguments. All functions in this section need the *aBase* license.

not

Return the logical not of the argument.

Syntax

(not *value*)

value

Can be anything

Description

Return the logical not of *value*. The not of #*f* and *nil* is #*t*. The not of any other value is #*f*.

boolean?

Predicate to check if an object is a boolean atom.

Syntax

(boolean? *value*)

value

Can be anything

Description

Return #*t* if the value is a boolean atom, #*f* otherwise.

and

Perform the logical and of the arguments.

Syntax

(and *obj1* *obj2* ... *objn*)

objn

Can be anything

Description

Evaluate the arguments from left to right. If the argument evaluates to `#for nil`, return the result immediately. Otherwise evaluate the next argument. If there are no more arguments, return the value of the last evaluation. If there are no arguments, return `#t`.

or

Perform the logical or of the arguments.

Syntax

`(or obj1 obj2 ... objn)`

`objn`

Can be anything

Description

Evaluate the arguments from left to right. If the argument evaluates to `#t` or non-*nil*, return the result immediately. Otherwise evaluate the next argument. If there are no more arguments, return the value of the last evaluation. If there are no arguments, return `#f`.

Equivalence Predicates

Equivalence predicates are described in the following sections.

eq?

Checks whether or not two objects are identical or equal.

or

Syntax

(eq? *obj1 obj2*)

where the arguments are:

obj1, obj2 These objects can be anything.

Description

Compare whether two objects are identical. This is the strongest test of equivalence. When the two arguments are identical, the function simply returns #t. When they are not, it will return #f.

Examples

(eq? 1 1) ; Returns #t because it is the exact same list or atom. -> #t

(define x 1)

(eq? x x) ; Returns #t because it is the exact same list or atom. -> #t

(eq? '() '()) ; Returns #t because it is identical. -> #t

(eq? #t #t) ; Returns #t because there is only one #t and #f atom. -> #t

eqv?

Check whether two objects are almost identical or equal.

Syntax

(eqv? *obj1 obj2*)

obj1, obj2

Can be anything

Description

Compare whether two objects are almost identical. This is the next strongest test of equivalence. If *eq?* would return #*t*, *eqv?* also returns #*t*. It also returns #*t* if the two atoms are equal.

Examples

```
(eqv? 1 1) ; Returns #t because the "1" atoms are equal.  
#t  
(define x '(1 2))  
(define y '(1 2))  
(eqv? x y) ; Returns #f because it is not the exact same list.  
#f  
(eqv? '() '()) ; Returns #t because they are identical.  
#t
```

equal?

Check if two objects are almost equal.

Syntax

(equal? *obj1* *obj2*)

obj1, obj2

Can be anything

Description

Compare if two objects are almost equivalent. This is the weakest test of equivalence. In general, if two objects print the same, *equal?* returns #*t*.

Examples

```
(define x '(1 2))
(define y '(1 2))
(equal? x y) ; Returns #t because each submember of the list
is equal.
#t
(equal? 1 2) ; Returns #f 1 != 2
#f
```

Pairs and Lists

Pairs are used to represent lists. A list can be either the empty list, which is specified by a set of parentheses (), or a pair whose *cdr* field is a list.

Note:

The empty list is not a pair. It has no elements and a length of zero.

Synopsys Scheme supports dotted notation for Scheme pairs. For example,

```
'(1 . 2)
```

pair?

Check if an object is a pair.

Syntax

```
(pair? obj)
```

obj

Can be anything

Description

Predicate to test if an object is a pair. If the object is a pair, #*t* is returned, else #*f*.

Examples

```
(pair? 1)  
⇒#f  
(pair? '(1 2))  
⇒#t
```

list?

Check if an object is a proper list.

Syntax

```
(list? obj)
```

obj

Can be anything

Description

Predicate to test if an object is a proper list. If the object is a proper list, #*t* is returned, else #*f*. A proper list is defined as a list with the last pair's *cdr* field containing *nil*.

Examples

```
(list? '())
  ↗#t
(list? '(1 . 2))
  ↗#f
(list? '(1 2))
  ↗#t
```

null?

Check if an object is the nil list.

Syntax

`(null? obj)`

obj

Can be anything

Description

Predicate to test if an object is the nil list. If the object is the nil list, `#t` is returned, else `#f`.

Examples

```
(null? 1)
  ↗#f
(null? '())
  ↗#t
```

cons

Prepend an object to a list

Syntax

(cons *obj1* *obj2*)

obj1,*2*

Can be anything

Description

Prepend *obj1* to *obj2*. The most general use of this function is to insert *obj2* as the first element of the list object *obj2*. The result of this operation is a new unique pair.

Examples

```
(cons 1 '(2 3))  
⇒(1 2 3)  
(cons '(1 2) 3)  
⇒((1 2) . 3)
```

car

Select and return the first element of the pair.

Syntax

(car *obj*)

obj

Must be a list

Description

Return the contents of the car field in the pair.

Examples

```
(car '(1 2))  
⇒1
```

cdr

Select and return the second element of the pair.

Syntax

(cdr *obj*)

obj

Must be a list

Description

Return the contents of the *cdr* field in the pair.

Examples

(cdr '(1 2))
⇒ (2)

set-car!

Set the first element (*car*) of a list object.

Syntax

(set-car! *obj1* *obj2*)

obj1

Must be a list.

obj2

Can be anything.

Description

Store *obj2* into the *car* field of the pair specified by *obj1*. The function returns the new value of pair *obj1*. Pair *obj1* is modified by this procedure.

Example

```
(set-car! ' (1 2) 3)  
⇒ (3 2)
```

set-cdr!

Set the second element (*cdr*) of a list object.

Syntax

```
(set-cdr! obj1 obj2)
```

obj1

Must be a list

obj2

Can be anything

Description

Store *obj2* into the *cdr* field of the pair specified by *obj1*. The function returns the new value of pair *obj1*. Pair *obj1* is modified by this procedure.

Examples

```
(set-cdr! ' (1 2) 3)  
⇒ (1 . 3)
```

length

Return the length of a list object.

Syntax

(length *obj*)

obj

Must be a list

Description

Return the length of the object *obj*. The length of the nil list is 0.

Examples

```
(length '(1 2))  
2  
(length '())  
0
```

append

Append the second list to the first.

Syntax

(append *obj1* *obj2*)

obj1

Must be a proper list

obj2

Must be a list (possibly improper)

Description

Create a new list composed of the elements of *obj1* followed by the elements of *obj2*. If *obj2* is not a proper list, the new list is improper. The new list shares the elements of *obj2*. Mutating *obj2* mutates the list returned by *append*. Mutating *obj1* does not mutate the list returned by *append*.

Examples

```
(append '(1 2) '(3 4))  
⇒(1 2 3 4)
```

Membership functions

The functions *memq*, *memv*, and *member* all test to see if an object is a member of a list.

Syntax

```
(memq obj1 obj2)  
(memv obj1 obj2)  
(member obj1 obj2)
```

obj1

Can be anything

obj2

Must be a list

Description

These three functions test whether *obj1* occurs as the *car* element of any of the sublists in *obj2*. If it does, the sublist is returned. If *obj1* is not an element, #*f* is returned. The three functions differ in the test that is used to determine whether *obj1* is equal. In the case of *memq*, the *eq?* test is used. In the case of *memv*, the *eqv?* test is used. In the case of *member*, the *equal?* test is used.

Examples

```
(memq 'a '(a b c))  
⇒(a b c)  
(member 1 '( 2 3 4))  
⇒#f
```

Bugs

These functions should test *obj2* to ensure it is a proper list. An error should be signalled if *obj2* is not a proper list. Currently no check is made.

Association functions

The functions *assq*, *assv*, and *assoc* all test to see if an object is a member of an association list.

Syntax

```
(assq obj1 obj2)  
(assv obj1 obj2)  
(assoc obj1 obj2)
```

obj1

Can be anything

obj2

Must be a list

Description

These three functions test if *obj1* occurs in the association list *obj2*. An association list is defined as a list of pairs. The key of each pair is the *car* of the pair. The value of each pair is the *cdr* of the pair. These procedures find the first pair in *obj2* whose *car* is equal to *obj1*. The test of equality is determined by which of the three functions is used. The *assq* function uses *eq?* for the test, *assv* uses *eqv?* for the test, and *assoc* uses *equal?* for the test. If no pair satisfies the test, *#f* is returned.

Examples

```
(assq 'a '((a 1) (b 2) (c 3) (a 4)))
⇒(a 1)
(assq 'v '((a 1) (b 2) (c 3) (a 4)))
⇒#f
```

Bugs

These functions should test *obj2* to ensure it is a proper association list. An error should be signalled if *obj2* is not a proper association list. Currently no check is made.

Symbol Predicates

Symbol predicates are described in the following sections.

symbol?

Check if an object is a symbol.

Syntax

```
(symbol? obj)
```

obj

Can be anything

Description

Predicate to test if an object is a symbol. If the object is a symbol, `#t` is returned, else `#f`.

Examples

```
(symbol? 1)  
⇒#f  
(symbol? 'x)  
⇒t
```

Numbers

Number Predicates

The `number?`, `real?`, `integer?`, `float?`, `zero?`, `positive?`, `negative?`, `odd?`, and `even?` functions all test an object for a characteristic.

Syntax

```
(number? obj)  
(real? obj)  
(integer? obj)  
(float? obj)  
(zero? num)  
(positive? num)  
(negative? num)  
(odd? int)  
(even? int)
```

obj

Can be anything

<i>num</i>	Must be a number (floating or integer)
<i>int</i>	Must be an integer

Description

The function *number?* tests *obj* if it is a number. A number is either an integer or a float object. The function *real?* is identical to *number?* in Synopsys Scheme. The functions *integer?* and *float?* test if the object is an integer object or float object respectively. The function *zero?* tests if the number is identical to zero. The functions *positive?* and *negative?* test if the number is greater than zero or less than zero, respectively. Zero is neither positive nor negative. The functions *odd?* and *even?* are restricted to integers and test if an integer is odd or even, respectively.

Number Equality Predicates

The *=*, *<*, *>*, *<=*, and *>=* functions all compare the magnitude of two numbers.

Syntax

```
(= num1 num2)
(< num1 num2)
(> num1 num2)
(<= num1 num2)
(>= num1 num2)
```

num1, *num2*

Must be a number (floating or integer).

Description

These functions test the relative magnitude of *num1* and *num2*. If *num1* is (=, <, >, <=, >=) than *num2*, #*t* is returned, else #*f*.

Arithmetic Functions

The functions *+*, *-*, ***, */*, *quotient*, *remainder*, and *modulo* provide basic arithmetic operations in Scheme.

Syntax

```
(+ num1 num2)
(- num1 num2)
(* num1 num2)
(/ num1 num2)
(quotient int1 int2)
(remainder int1 int2)
(modulo int1 int2)
```

num1, *num2*

Must be a number (floating or integer)

int1, *int2*

Must be an integer

Description

The functions *+*, *-*, *** and */* return the sum, difference, product, and quotient of their operands, respectively. If both operands are integers, the result is an integer. The exception to this is the */* operator. If the result of the division is an integer, an integer value is returned, else a float. The function *quotient* divides *int1* by *int2* using integer division rules. The result is an integer. The sign of the result is the sign of the product of the two operands. The *modulo* and *remainder* functions return the same result if both operands are positive. The result is (*int1* - (*quotient* *int1* *int2*)). For the remainder

function, where the operands are not both positive, take the absolute value of both operands, execute the computation, and apply the sign of the *int1* operand to the result. For operands that have mixed signs, see the examples.

Examples

```
(modulo 13 4)
ž1
(modulo -13 4)
ž3
(modulo 13 -4)
ž-3
(modulo -13 -4)
ž-1
(remainder 13 4)
1
(remainder -13 4)
-1
(remainder 13 -4)
-1
(remainder -13 -4)
-1
```

Number Conversion

The functions *floor*, *ceiling*, *truncate* and *round* all convert floating point numbers to integers with various methods.

Syntax

```
(floor num)
(ceiling num)
(truncate num)
(round num)
```

num

Must be a number (floating or integer).

Description

The function *floor* returns the largest integer not greater than the operand. The function *ceiling* returns the smallest integer not less than the operand. The function *truncate* returns the integer part of the operand. The function *round* rounds the operand to the nearest integer according to the IEEE rounding direction.

char?

Predicate to test if an object is a character.

Syntax

(char? *obj*)

obj

Can be anything

Description

Return #*t* if the object is a character, #*f* otherwise.

char->integer

Convert a character object into its ASCII numeric equivalent.

Syntax

(char->integer *obj*)

obj

Must be a character

Description

Convert a character object into its ASCII numeric equivalent.

integer->char

Convert an integer into a character object.

Syntax

(integer->char *obj*)

obj

Must be a positive integer less than 256

Description

Convert an integer into a character object.

char-upcase

Convert a character to its upper case equivalent.

Syntax

(char-upcase *obj*)

obj

Must be a character

Description

Convert a character to its upper case equivalent. If the character is already uppercase, or if the object is a non-alphabetic character, the result is *obj*.

char-downcase

Convert a character to its lower case equivalent.

Syntax

(char-downcase *obj*)

obj

Must be a character

Description

Convert a character to its lower case equivalent. If the character is already lowercase, or if the object is a non-alphabetic character, the result is *obj*.

String Functions

String functions are described in the following sections.

string?

Predicate to test if an object is a string.

Syntax

(string? *obj*)

obj

Can be anything

Description

Return #*t* if the object is a string, #*f* otherwise.

string-length

Return the number of characters in a string.

Syntax

(string-length *obj*)

obj

Must be a string

Description

Return the number of characters in a string.

string-ref

Return a specific character inside a string.

Syntax

(string-ref *obj* *k*)

obj

Must be a string

k

Must be an integer

Description

Return the *k*th character in string *obj*. Zero based indexing is used. It is an error to try to index a character outside of the length of the string.

string-set!

Set a specific character inside a string.

Syntax

(string-set! *obj* *k* *c*)

obj

Must be a string

k

Must be an integer

c

Must be a character

Description

Set the *k*th character in string *obj* to *c*. Zero-based indexing is used. It is an error to try to set a character outside of the length of the string.

substring

Return a specific substring of a string.

Syntax

(substring *obj* *k* *l*)

obj

Must be a string

k

Must be an integer

/

Must be an integer

Description

Return a substring bounded by indexes *k* and *l*. Zero-based indexing is used. The index *k* must be smaller than or equal to *l*. The indexes *k* and *l* must be bounded by zero and the length of the string. The substring must include the *k*th element and must not contain the *l*th element. If *k* and *l* are equal, a zero-length string is returned.

string-append

Concatenate two strings.

Syntax

(string-append *obj1* *obj2*)

obj1 *obj2*

Must be a string

Description

Append string *obj2* to string *obj1*.

string->list

Convert a string into a list of characters.

Syntax

(string->list *obj*)

obj

Must be a string

Description

Convert the string *obj* into a list of characters. If the string is of zero length, *nil* is returned.

list->string

Convert a list of characters into a string.

Syntax

(list->string *obj*)

obj

Must be a proper list of characters

Description

Convert the list of characters to a string object.

string-copy

Create a copy of a string.

Syntax

(string-copy *obj*)

obj

Must be a string

Description

Create a copy of a string. Changing the new string will not affect the original string.

string-fill!

Fill a string with a character.

Syntax

(string-fill! *obj* *c*)

obj

Must be a string

c

Must be a character

Description

Replace all of the elements of the string with the character *c*. Returns the modified string.

string->float

Convert a string to a float object.

Syntax

(string->float *obj*)

obj

Must be a string

Description

Convert the string object *obj* to a floating-point number. If the string does not contain a number, 0.0 is returned. Note that this is a nonstandard function and replaces the standard function *string->number*.

string->integer

Convert a string to an integer object.

Syntax

(string->integer *obj*)

obj

Must be a string

Description

Convert the string object *obj* to an integer number. If the string does not contain a number, 0 is returned. Note that this is a nonstandard function and replaces the standard function *string->number*.

number->string

Convert a number to an equivalent string representation.

Syntax

(number->string *obj*)

obj

Must be a number

Description

Convert the number object to a string representation. If the object is an integer, no decimal point is printed. If it is a float, the format is unspecified.

Control Features

Control features are described in the following sections.

procedure?

Predicate to test if an object is a procedure.

Syntax

(procedure? *obj*)

obj

Can be anything

Description

Return #t if the object is a procedure, #f otherwise.

apply

Apply a procedure to arguments.

Syntax

(apply *proc args*)

proc

Must be a procedure

args

Must be a list

Description

Apply procedure *proc* using *args* as the operands.

Example

```
(apply + '(1 2))  
3
```

map

Map a procedure against multiple sets of operands, returning the result as a list.

Syntax

```
(map proc args)  
(map proc list1 list2 ... listn)
```

proc

Must be a procedure

args, list

Must be a list

Description

In the first form, the length of *args* must be a multiple of the number of operands expected by *proc*. In this form, the first *n* elements of *args* are used as a set of operands to *proc*. These sets of operands

are evaluated by *proc* in some unspecified order. The results of the evaluations are bundled into a result list. (This is the Revision 3 specification.)

The second form takes one element from each list and applies the procedure *proc* to this set of operands. All of the lists must be of the same length. As in the first form, the results are bundled into a list and returned.

Example

```
(map + ' (1 2 3 4))  
(3 7)  
(map + ' (1 3) ' (2 4))  
(3 7)
```

Bugs

Because both Revision 3 and Revision 4 are supported, if a procedure which is defined to take a variable number of arguments and the first form is specified, the template for the second form will be used. The Revision 3 method would have signalled an error in this case.

for-each

Apply a procedure iteratively over a set of arguments.

Syntax

```
(for-each proc args)  
(for-each proc list1 list2 ... listn)
```

proc

Must be a procedure

args, list

Must be a list

Description

In the first form, the length of *args* must be a multiple of the number of operands expected by *proc*. In this form, the first *n* elements of *args* are used as a set of operands to *proc*, the next *n* *args* and so on. These sets of operands are evaluated by *proc* in order. The result of *for-each* is always *nil*. (This is the Revision 3 specification).

The second form takes one element from each list and applies the procedure *proc* to this set of operands. All of the lists must be of the same length. As in the first form, *nil* is returned.

Bugs

Because both Revision 3 and Revision 4 are supported, if a procedure which is defined to take a variable number of arguments and the first form is specified, the template for the second form will be used. The Revision 3 method would have signalled an error in this case.

call-with-current-continuation

Creates a procedure that allows for non-local returns.

Syntax

(call-with-current-continuation *proc*)

proc

Must be a procedure of exactly one argument

Description

Provides an equivalent to the C `setjmp/longjmp` technique. The *proc* procedure is executed normally unless a call to its argument is made. If such a call is made, *call-with-current-continuation* returns immediately, with a value specified by the argument to *proc*. This implementation of *call-with-current-continuation* is less general than the Scheme Revision 4 report specifies. The extent of the defined procedure *proc* is limited to the context of the *call-with-current-continuation* body. The report allows for unlimited extent for *proc*.

The most common use of *call-with-current-continuation* is to “break” from loops or to stop a recursive function prematurely. The procedure created by *call-with-current-continuation* is a special type, and has a unique atom associated with it.

Example

```
(call-with-current-continuation
  (lambda (rtn)
    (for-each (lambda (x) (if (zero? x) (rtn x)))
      '(1 2 3 0 4 5 6))
    #t))
  0
```

In this example, the *for-each* function is terminated when the 0 element of the list is operated on. The value 0 is returned immediately. If no 0s were specified in the list, the result of the expression would be simply #t. Note that the newly created function *rtn* as a result of the *call-with-current-continuation*, expects exactly one argument.

Input and Output

input-port?

Predicate to test if an object is an input type port.

Syntax

(input-port? *obj*)

obj

Can be anything

Description

Return #*t* if the object is an input port, #*f* otherwise.

output-port?

Predicate to test if an object is an output type port.

Syntax

(output-port? *obj*)

obj

Can be anything

Description

Return #*t* if the object is an output port, #*f* otherwise.

open-input-file

Open a file for reading and return a port object.

Syntax

(open-input-file *obj*)

obj

Must be a string

Description

This function, along with *read-char* and *read-string*, forms the basis for reading files in Scheme. The *obj* operand should be the name of the file to be opened. The file must exist, and the file must have at least read permission. You need not specify the path if the file is in the current directory. If the file is not in the current directory, you can specify either an absolute or a relative path. Leading ~ prefixes are properly handled. Ports are printed as *p:<fileName>*. If a port is garbage collected, it is closed as a side effect. Take care to close ports when you are done using them. Each port requires one file descriptor, and the system is normally limited to 64 file descriptors. Severe errors can occur if the system exhausts the number of file descriptors available.

Example

```
(open-input-file "~guest/afile")
p:/mnt/guest/afile
```

open-output-file

Open a file for writing and return a port object.

Syntax

(open-output-file *obj*)

obj

Must be a string

Description

This function, along with *display* and *print*, forms the basis for writing files in Scheme. The *obj* operand should be the name of the file to be opened. The file must be writable. You need not specify the path if the file is in the current directory. If the file is not in the current directory, you can specify either an absolute or a relative path. Leading ~ prefixes are properly handled. Ports are printed as *p:<fileName>*. If a port is garbage collected, it is closed as a side effect. Take care to close ports when you are done using them. Each port requires one file descriptor, and the system is normally limited to 64 file descriptors. Severe errors can occur if the system exhausts the number of file descriptors available.

Example

```
(open-output-file "~guest/afile")
p:/mnt/guest/afile
```

close-input-port

Close a port that was open for read.

Syntax

(close-input-port *obj*)

obj

Must be an input-port

Description

Close the port, freeing the associated file descriptor. Further attempts to read from this port will signal an error.

close-output-port

Close a port that was open for write.

Syntax

(close-output-port *obj*)

obj

Must be an output-port

Description

Close the port, freeing the associated file descriptor. Further attempts to write to this port will signal an error.

read-char

Read a character from a port.

Syntax

(read-char *port*)

port

Must be an input-port

Description

This procedure returns the next character available from the input-port and updates the *port* to point to the following character. When no more characters are available, *read-char* returns the EOF character.

If *port* is omitted, this function treats the default port as the “current input port,” which is the standard input from the console in most cases. If there is no input port on the console, Astro freezes until it gets user input from the console.

eof-object?

Predicate to test if a character is the special end of file character.

Syntax

(eof-object? *obj*)

obj

Must be a character

Description

Return #*t* if the object is the special end of file character, #*f* otherwise.

read-string

Read a string from a port.

Syntax

(read-string *port*)

port

Must be an input-port

Description

This procedure returns the next available string from input-port and updates *port* to point to the following string. A string is defined as a set of characters delimited by white space. When another string is not available, *read-string* returns '(), signalling end of file. Note that *read-string* is not part of the Revision 4 standard.

If *port* is omitted, this function treats the default port as the “current input port,” which is the standard input from the console in most cases. If there is no input port on the console, Astro freezes until it gets user input from the console.

display

Write an object to an output port.

Syntax

(display *obj* [*port*])

obj

Can be anything

port

Must be an output-port

Description

This procedure writes *obj* in an ASCII format. If *port* is specified, the output is sent to the specified port. If no port is specified, the output is displayed in the message area.

newline

Write a newline to the specified port.

Syntax

(newline [port])

port

Must be an output-port

Description

This procedure writes a newline character. If *port* is specified, the newline is sent to the specified port. If no port is specified, the newline is displayed in the message area.

write-char

Write a character to an output port.

Syntax

(write-char *obj* [port])

obj

Must be a character

port

Must be an output-port

Description

This procedure writes the character *obj* in an ASCII format. If *port* is specified, the character is sent to the specified port. If no port is specified, the character is displayed in the message area.

System Interface

load

Load a file to the evaluator.

Syntax

(load *fileName*)

fileName

Must be a string

Description

This procedure reads expressions and definitions from *fileName* and sequentially evaluates the expressions and definitions. The *fileName* argument should be a string that names an existing file containing Scheme source code. The file must exist, and the file must have at least read permission. You do not need to specify the path if the file is in the current directory. If the file is not in the current directory, you can specify either an absolute or a relative path. Leading ~ prefixes are properly handled. The *load* function returns #*t* upon success and returns *nil* if an error occurred during the load.

Loads fail if there are no more file descriptors available. Each *load* function takes one file descriptor. The operating system typically limits a process to 64 file descriptors, of which the program normally consumes 20. This implies that there is a limit to the number of nested loads. That limit varies depending on what other file descriptors are opened by the tool. Take care to limit the number of nested loads, as too many can adversely affect the program. If all file descriptors are exhausted by nested loads, serious program errors can occur.

3

General Extensions

This chapter provides descriptions and syntax for general extensions to the Scheme language. These extensions enhance the language's ability to interact with the operating system and provide additional numeric processing features.

Note:

Unless otherwise indicated, the examples provided are from the Astro application.

bit-and

Performs a logical AND on two integers.

Syntax

(bit-and *integer1 integer2*)

Examples

```
bit-and 1 2
0
bit-and 1 1
1
```

bit-or

Performs a logical OR on two integers.

Syntax

(bit-or *integer1 integer2*)

Examples

```
bit-or 1 2
3
bit-or 1 1
1
```

bit-not

Performs a logical NOT on an integer.

Syntax

(bit-not *number*)

Example

```
bit-not 1  
-2
```

CONSTANT

Lists the value(s) for either a specified constant or all constants.

Syntax

```
(CONSTANT "constantName")
```

where the argument *constantName* can be any of the following:

- Name of the constant of whose current value you want. Valid values are the names of any program constant. If you do not know the exact name, use "" to list the values of all constants.
- "", which lists the values of all constants.
- "*db-null*," which is generally used in testing the fields of *dbFetchObject*.

Example

```
CONSTANT ""  
INT_MAX 2147483647  
INT_MIN -2147483648  
dbcRotate0 0  
dbcRotate90 2  
dbcRotate180 4  
dbcRotate270 6  
dbcRotate0MirrorX 12  
dbcRotate90MirrorX 10  
dbcRotate180MirrorX 8  
dbcRotate270MirrorX 14  
dbcRotate0MirrorY 8  
dbcRotate90MirrorY 14  
dbcRotate180MirrorY 12  
dbcRotate270MirrorY 10  
. . .  
dbbNetSubTypeSynNetRoot 8  
dbbNetSubTypeSynNet 16  
db-null db:0
```

This example lists the current values of all program constants.

License Required

aBase

See Also

[“dbFetchObject” on page 4-82](#)

FileExists?

Test if the specified file exists.

Syntax

(FileExists? *fileName*)

fileName

Must be a string

Description

FileExists? tests if the directories leading to the specified file can be searched, and the file exists. The function returns #t if the directories are searchable and the file exists. If the directories cannot be searched or the file does not exist, an error code, which is an integer, will be returned. The error codes are taken from the system error codes defined in system include file error.h.

License Required

aBase

FileIsReadable?

Test if the specified file can be opened for read.

Syntax

(FileIsReadable? *fileName*)

fileName

Must be a string

Description

FileIsReadable? tests if the specified file can be opened for read, and the file exists. The function returns #t if the file can be opened for read. If the file cannot be opened for read or the file does not exist, an error code, which is an integer, will be returned. The error codes are taken from the system error codes defined in system include file error.h.

License Required

aBase

FileIsWritable?

Test if the specified file can be opened for write.

Syntax

(FileIsWritable? *fileName*)

fileName

Must be a string

Description

FileIsWritable? tests if the specified file can be opened for write, and the file exists. The function returns #t if the file can be opened for write. If the file cannot be opened for write or the file does not exist, an error code, which is an integer, will be returned. The error codes are taken from the system error codes defined in system include file error.h.

License Required

aBase

FileIsExecutable?

Test if the specified file can be executed.

Syntax

(FileIsExecutable? *fileName*)

fileName

Must be a string

Description

FileIsExecutable? tests if the specified file can be executed, and the file exists. The function returns #t if the file can be executed. If the file cannot be executed or the file does not exist, an error code, which is an integer, will be returned. The error codes are taken from the system error codes defined in system include file error.h.

Note that this function only tests if the file has execute permission. It does not test if the file is a valid executable file.

License Required

aBase

getDisplayStyle

To change the display modes, use “[setDisplayStyle](#)” on page 3-19.

Example

```
getDisplayStyle --> "QueryBoxes" #t  
"StopOnFormFieldError"  
#f
```

This example returns a list of all the enabled display modes.

License Required

Milkyway

See Also

[“setDisplayMode” on page 3-19](#)

getenv

Returns the current setting of a UNIX environment variable.

Syntax

```
(getenv "UNIX_variable")
```

where *UNIX_variable* is any string corresponding to a valid UNIX environment variable. For example,

- HOME
- DISPLAY
- WINDOWID

Note:

To list the current settings for all environment variables, type the following at the UNIX prompt:

```
env
```

Examples

```
getenv "HOME"  
"/mnt/jives"
```

The following example illustrates the usefulness of *getenv* in file name construction.

```
load (string-append (getenv "HOME") ".arcsrc")
```

For more information about the Synopsys implementation of standard Scheme commands in this example, see “[load](#)” on page 2-61 and “[string-append](#)” on page 2-45.

getInstallPath

Returns the path to where the Synopsys application is installed.

Syntax

```
(getInstallPath)
```

Examples

```
getInstallPath  
"/mnt/jives/devHier/"
```

The following example illustrates the usefulness of *getInstallPath* in path name construction.

```
loadModule "dynApplInitSCSymTbl" (list (string-append  
(getInstallPath) "dynAppl/dynAddIOText.o"))
```

In this example, *getInstallPath* builds a path name to the file *dynAddIOText.o* in the install directory. For more information, see “[loadModule](#)” on page 3-14.

getIntegVersion

Returns the integration version number of the program.

Syntax

```
(getIntegVersion)
```

See Also

- “getVersionIdent” on page 3-13
- “getVersion” on page 3-12
- “getSubVersion” on page 3-11
- “getProgramName” on page 3-11

getInterruptMask

Returns the current interrupt mask. See “[Interrupts](#)” on page 2-11 for more information.

Syntax

`(getInterruptMask)`

getInterruptPollInterval

Returns the current interrupt polling interval. See “[Interrupts](#)” on page 2-11 for more information.

Syntax

`(getInterruptPollInterval)`

getProgramName

Returns the name of the program from which the function is issued.

Syntax

```
(getProgramName)
```

Example 1

```
getProgramName  
"Astro"
```

Example - Load *.arcsrc* if Astro is currently running.

```
if (equal? (getProgramName) "Astro")  
(load (string-append (getenv "HOME") "/.arcsrc"))
```

In this example, \$HOME/.arcsrc loads only if Astro is the current program running.

See Also

[“getVersionIdent” on page 3-13](#)

[“getVersion” on page 3-12](#)

[“getSubVersion” on page 3-11](#)

[“getIntegVersion” on page 3-9](#)

getSubVersion

Returns the subversion number of the program.

Syntax

```
(getSubVersion)
```

See Also

- [“getVersionIdent” on page 3-13](#)
 - [“getVersion” on page 3-12](#)
 - [“getIntegVersion” on page 3-9](#)
 - [“getProgramName” on page 3-11](#)
-

getVersion

Returns the version number of the program.

Syntax

```
(getVersion)
```

Example

```
getVersion  
2.3
```

Example - AstroGA

```
getVersion  
2.0
```

See Also

- [“getVersionIdent” on page 3-13](#)
- [“getSubVersion” on page 3-11](#)
- [“getIntegVersion” on page 3-9](#)
- [“getProgramName” on page 3-11](#)

getVersionIdent

Returns complete information about the version of the program.

Syntax

```
(getVersionIdent)
```

Example

```
getVersionIdent
"Astro version 2.3.1.39 Sun Jul 24 09:22:44 PDT 1994
(arcs9)"
```

Example - AstroGA

```
getVersionIdent
"AstroGA version 1.1.0.39 Sat Jul 23 21:26:25 PDT
1994
(arcs9)"
```

See Also

[“getVersion” on page 3-12](#)

[“getSubVersion” on page 3-11](#)

[“getIntegVersion” on page 3-9](#)

[“getProgramName” on page 3-11](#)

getwd

Prints the working directory.

Syntax

```
(getwd)
```

Example

```
getwd  
"/usr1/amber/training/example1"
```

interrupt?

Alerts the user that an interrupt has occurred since the beginning of the last top-level evaluation. The *interrupt?* function returns #f if no interrupt has been signaled. If the Synopsys application detects an interrupt, *interrupt?* returns #t. The interrupt flag is not reset until the end of the current top-level evaluation. See “[Interrupts](#)” on page 2-11 for more information.

Syntax

```
(interrupt?)
```

See Also

“[signal-interrupt-error](#)” on page 3-22 contains an example of *interrupt?*

loadModule

Loads the object code compiled from external C programs into the currently executing program.

Syntax

```
(loadModule "CFunction" objectFiles)
```

where the arguments are as follows:

CFunction

Indicates the C function you immediately want to call after loading the object code.

Valid values: Name of any C function

objectFiles

Name(s) of object file(s) you want to load

Valid syntax:

'({ "filename" } ...)

Example

```
loadModule "initFunction"
'("myProg.o" "myProgAux1.o" "myProgAux2.o")
#t
```

License Required

aBase (with ABDynLnK)

See Also

[“getInstallPath” on page 3-9](#) for another example of *loadModule*

[“queryModules” on page 3-16](#)

[“releaseModule” on page 3-18](#)

loadModuleBindImmediate

Specifies how symbol references are resolved when the library is already loaded. This command should be executed before you run the “[loadModule](#)” on page 3-14 command.

Syntax

```
loadModuleBindImmediate setFlag
```

where the arguments are as follows:

setFlag	#t indicates that the symbol reference will be resolved immediately when the library is loaded. #f indicates that the symbol resolution will be delayed until the actual reference.
---------	--

Example

```
loadModuleBindImmediate #t  
#t
```

License Required

Milkyway, MDynLink

See Also

[“loadModule” on page 3-14](#)

queryModules

Returns a list of dynamically linked modules that are currently loaded.

Syntax

(queryModules)

Example

```
queryModules  
("myProg.o" "myProgAux1.o" "myProgAux2.o") STABIE
```

License Required

aBase (with ABDynK)

See Also

[“loadModule” on page 3-14](#)

[“releaseModule” on page 3-18](#)

rand

Generates successive pseudo-random numbers between 0 and (231)-1. Initially, *rand* is seeded with a value of 1. To reset the random-number generator to a random starting point, use [“srand” on page 3-24](#).

Syntax

(rand)

License Required

aBase

See Also

[“srand” on page 3-24](#)

releaseModule

Releases loaded dynamically linked modules.

To display a list of dynamically linked modules, use “[queryModules](#)” on page 3-16.

Syntax

`(releaseModule moduleNo)`

where *moduleNo* is an integer that the Synopsys application assigns to the dynamically linked module during loading.

Note:

This integer, based on the order in which the Synopsys application loads the module(s), is not persistent.

License Required

aBase (with ABDynK)

See Also

[“loadModule” on page 3-14](#)

[“queryModules” on page 3-16](#)

dbSetBckFile

Flags the database to create a backup file.

Syntax

`(dbSetBckFile boolean)`

where the argument is as follows:

boolean

Whether or not a backup file should be created

Valid values:

- *#t* asks the database to create a backup file.
- *#f* asks the database to not create a backup file.

License Required

aBase

setDisplayMode

Use this command to set the display mode of the second query box (YesNoDialogBox) and the error message box (OnFormFieldError).

Syntax

(setDisplayMode "type" mode)

where the arguments are as follows:

type Name of system indicator to set.

Valid values:

- QueryBoxes
- StopOnFormFieldError

mode Indicates whether the system indicator operates or not.

Valid values:

- *#t* enables the mode.
- *#f* disables the mode.

Type the options: “QueryBoxes” “StopOnFormFieldError”;

Example

1. setDisplayMode "QueryBoxes" #f

Sets the nullDisplay mode of your Synopsys session. If the result is TRUE, all occurrences of YesNoDialogBox will be disabled.

2. setDisplayMode "StopOnFormFieldError" #t

Sets the mode so that the replay file does not stop on a form/field mismatch or error.

To list the enabled display modes, use “[getDisplayStyle](#)” on [page 3-7](#).

License Required

Milkyway

setInterruptMask

Allows user control of the interrupt mask. See “[Interrupts](#)” on [page 2-11](#) for more information.

Syntax

(setInterruptMask *integer*)

integer

Must be an integer

License Required

aBase

setInterruptPollInterval

Allows user control of the interrupt poll interval. See “[Interrupts](#)” on [page 2-11](#) for more information.

Syntax

`(setInterruptPollInterval integer)`

integer

Must be an integer greater than 0

License Required

aBase

dbSetPriv

Allows the system to print (or not to print) privilege information under *dbQuery*.

Syntax

`(dbSetPriv boolean)`

where the argument is as follows:

boolean

Specifies whether or not privilege information should print under *dbQuery*

Valid values:

- *#t* prints privilege information.
- *#f* prohibits printing of privilege information.

License Required

aBase

See Also

[“dbQuery” on page 4-107](#)

[“dbOpenQueryFile” on page 4-106](#)

signal-interrupt-error

Prints an interrupt error message. Upon evaluation of *signal-interrupt-error*, the evaluator returns the user to the top level.

Syntax

(signal-interrupt-error)

Example

```
define displayObjects (lambda (type)
define cellId (geGetEditCell)
(db-foreach cellId '() type z
(begin
(print (dbFetchObject cellId z))
(if (interrupt?) (signal-interrupt-error)
#t)
)
)
)
(sfunc)
displayObjects "net"
(("netname" "IOPAD1") ("numOfMembers" 3)
("netHeaderId" db:0) ("childNetHeaderId" db:0)
("propertyHeaderId" db:0)
("cellRegionNumber" 0)
.
.
.
^C
ERROR:User Interrupt signaled.
Stack follows
Signal-interrupt-error.if.begin.db-foreach.UserProc.
()
```

In this example, the user-defined function, *displayObjects*, displays all objects of net type. While this command is being executed, *interrupt?* determines whether *displayObjects* is interrupted. The *signal-interrupt-error* function signals an interrupt error when you type CTRL+C.

License Required

aBase

See Also

[“interrupt?” on page 3-14](#)

sleep

Puts the program to sleep *ntime* seconds.

Syntax

`(sleep ntime)`

where *ntime* is the increment of time the program waits before resuming execution. The argument *ntime* can be any positive integer.

strand

Resets the random-number generator to a specific seed value.

Syntax

`(strand seed)`

seed

Must be an integer

License Required

aBase

See Also

[“rand” on page 3-17](#)

system

Executes shell commands from within the program. The program waits for the completion of the spawned process. The result is the result of the “C” function system.

Note:

The stdIn and stdOut ports are not available. Instead, use the redirect operators <, >, and >>.

Syntax

```
(system "UNIXcommand")
```

Example 1

```
system "awk -f program.awk inputFile > outputFile"
```

Examples 2 and 3 illustrate how to escape quotes within the UNIX shell command.

Example 2

```
system "echo '' >> ex4.clf"
```

Example 3

```
system "echo 'definePad \"*\" \"90\"' >> ex4.clf"
```

time

Displays the amount of time passed since the invention of UNIX, 00:00:00 GMT, Jan. 1, 1970, measured in seconds.

Syntax

```
(time)
```

Example

time
773023409

License Required

aBase

4

Scheme Database Extensions

This chapter presents basic information about the Synopsys database, and how Scheme has been extended to access the database. In addition, database schema information is presented in this chapter.

Understanding the Synopsys Database

The Synopsys database consists of a collection of records referenced by unique object IDs. In the Synopsys database, there are three basic types of object IDs:

- Library IDs, which are not persistent and not context sensitive. The ID is sufficient and unique by itself.
- Cell IDs, which are not persistent and not context sensitive. The ID is sufficient and unique by itself.
- Other IDs, which are persistent and require the context of either a cell ID or a library ID to be understood by the database. Other IDs require either a cell ID or library ID because more than one cell or library may have an object with the same database ID.

When a library or cell is opened, the Synopsys application assigns it a unique, temporary database ID that is dependent upon the order in which the library or cell is accessed. After the session, the Synopsys application does not store this ID for future use.

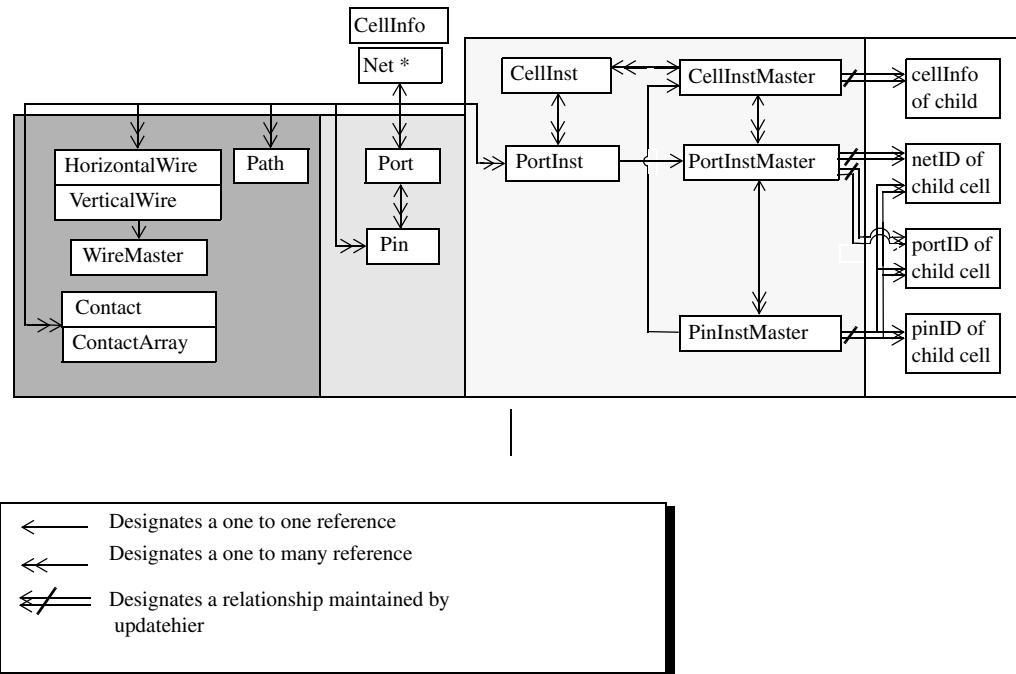
Other IDs, unlike library or cell IDs, are persistent. During the creation of an object, the Synopsys application assigns a database ID, which is a permanent pointer to the object. The persistence of other IDs assures that if port A of cell TOP has a database ID db:100004 in session 1 of Astro, port A will continue to have that ID in future sessions.

Under certain modification operations, an ID may change. Further, if an object is deleted, its ID may be reused by the database. Therefore, you should be careful in assuming the persistence of an ID.

References Between Database Objects

Some types of objects can have relations between themselves and other objects. [Figure 4-6](#) illustrates the relationships between database objects. The following legend differentiates the various types of relationships existing between database objects.

Figure 4-6 Database Object Relationships



Naming Guidelines

This section describes the Synopsys database naming restrictions. If you plan to interface with other tools or export data to other formats, you need to also adhere to the guidelines for those tools or formats.

Character restrictions

For most database objects, names can contain any characters. For libraries and cells, names are restricted to the following characters:

- Alphabetic characters
- Numeric characters
- Underscores (_)
- Dollar signs (\$)
- Pipe signs (|)

Cell names can also contain the following special-function characters, which are added by your Synopsys tool:

- Periods (.) separate cell names from their extensions, which indicate the type of the cell. For example, *cellA.cel* indicates that the *cellA* is a layout cell.
- Semicolons (;) separate cell names (including extensions) from their version numbers. For example, *cellA.cel;2* indicates version 2 of *cellA.cel*.

Length Limitations

The following table lists Synopsys objects and the name length limitations for each.

Table 4-3 Name Length Limitations

Object	Name length limitation
application	31 characters
cell	128
cell instance	1023 characters

Table 4-3 Name Length Limitations (Continued)

Object	Name length limitation
color (defined in technology file)	31 characters
contact (defined in technology file)	31 characters
error class	15 characters
group	31 characters
layer	31 characters
library	255 characters
line style (defined in technology file)	15 characters
mask layer (defined in technology file)	31 characters
net	1023 characters
pad	31 characters
pin	31 characters
port	1023 characters
property	35 characters
reference library	127 characters
region pin	31 characters
stipple pattern (defined in technology file)	15 characters
technology file	31 characters
tile array class (defined in technology file)	31 characters
tile array class subtile (defined in technology file)	31 characters
units (defined in technology file)	31 characters

Case Sensitivity

Case sensitivity refers to the recognition of the case of letters. In a case-sensitive system, uppercase and lowercase letters are differentiated; while in a case-insensitive system, they are not. For example, in a case-sensitive system, *xyz* and *XyZ* refer to two different objects; in a case-insensitive system, these two names refer to the same object.

Your Synopsys tool is restricted to case-sensitivity for the following:

- Library directory and name
- Object type name
- Attribute name

Your Synopsys tool allows case-insensitivity for the following:

- Tech-File
 - Color Name
 - Line-Style Name
 - Stipple Name
- replay file
 - Form Name
 - Field Name

For the following, your Synopsys tool bases case-sensitivity on the library content case sensitivity:

- Property name

- Attach file attach name
- View name (directory name)
- Cell name
- Object group name

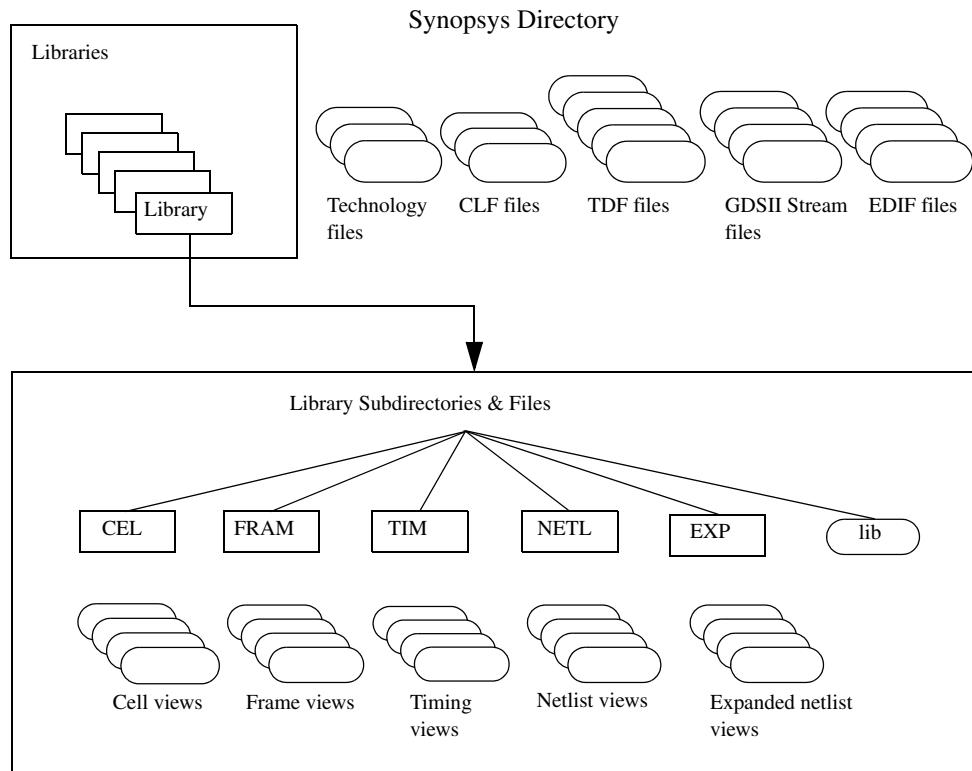
To start a Synopsys tool in case-sensitive mode, use the `-caseSensitive` command line option (see “Command Line Options” in *Physical Implementation Getting Started*). To verify whether your current session default mode is case sensitive, use `getCaseSensitive` (see the *I/C Implementation Command Reference Manual*). To switch the default case-sensitivity mode after you start a Synopsys tool (but before you open a library), use `setCaseSensitive` (see the *I/C Implementation Command Reference Manual*).

- `#t` indicates that you want to wait (generally used in scripts to indicate that you want to suspend execution of the script until a license becomes available).
- `#f` indicates that you do not want to wait.

Working with Cell Libraries

The following figure provides a graphical representation of the data structures that make up your Synopsys database.

Figure 4-7 Synopsys File Structure



Cell Libraries

A cell library can contain both top-level designs and the cells used in those designs. In some cases, cells in a library can contain cells from other libraries. Instead of copying cells to multiple libraries, a Synopsys application references the other libraries, which are called “reference libraries.”

Your Synopsys directory should contain the following for each library *<libraryName>*:

- Library subdirectories containing each cell type in the library. See “Working with Cells” on page 3-54.

- A library information file (*lib*). This is a binary file that contains information compiled from a technology file, a cell library format (CLF) file, and the cell files inside the library directory.

Binary files cannot be edited. However, you can change the technology and CLF files by using Milkyway to unload and reload these files. When unloading one of these files, Milkyway places the information in an editable format so that you can modify it using a UNIX text editor.

Synopsys provides sample libraries, which you can find in the directory where you installed your Synopsys application. From the install directory, change to the *demo* directory and look for these files.

Naming Libraries

Guidelines for naming libraries include the following:

- The first character should be an alphabetic character.
- The remaining characters may be any of the following:
 - Any alphabetic characters
 - Any numeric characters
 - Underscore (_)
- The length should not exceed 127 characters.

Accessing Libraries

You can open a library in either of the following modes:

- Read-only mode lets you open the library (read) but not make any changes to it (write).
- Write mode lets you open the library and make changes (read and write).

Note:

Synopsys tools can have only one library open in write mode.

When you open a cell library, the Synopsys application assigns a database object ID to it. (Note that this ID is valid only until the library is closed or the Synopsys session ends.) Some commands need this library ID as an argument. Since you can only open one library at a time for write, dbOpenLib (see page 3-39 of the *I/C Implementation Command Reference Manual*) globally specifies the library ID. Most other Scheme extensions assume this global library ID.

Specifying Libraries

To specify a library ID, bind the ID to a variable when you open a cell library and use the variable for *libId*. For example,

```
define lib1 (dbOpenLib "test")
db:1
dbGetLayerNumberFromName lib1 "METAL1"
9
```

In this example,

- The first line opens the library and creates a top-level definition (*lib1*) for the database object ID of the library.

- The second line scans the library for the layer number of *METAL 1* and returns its layer number.

Synopsys provides another extension for opening libraries, *dbOpenLibrary* (see page 3-40). However, you are strongly advised not to use this extension when opening a library in write (W) mode. Unlike *dbOpenLib*, *dbOpenLibrary* does not globally specify the database object ID of a library.

If the library ID is not bound to a variable or the variable is lost, you can do one of the following:

- Use *dbGetCurrentLibId* as follows:

```
define identifier (dbGetCurrentLibId)
```

where the argument *identifier* is the name of the variable to which the library ID binds.

For example,

```
define lib1 (dbGetCurrentLibId)
db:1
```

- If a cell is open in your library, use *dbGetLibIdByCellId* (see page 3-116) as follows:

```
define identifier (dbGetLibIdByCellId cellId)
```

where the arguments are as follows:

identifier

Name of the variable to which the library ID binds

Valid values: Any text string

cellId

Database object ID of any open cell in the library for which you want to find the library ID

Valid values:

- Cell ID
- (`geGetEditCell`)

Note:

You can only use this value if you opened the cell using the *Cell > Open* menu command, `geOpenCell` (see the *IC Implementation Command Reference Manual*). The `geGetEditCell` function returns the cell ID of the cell in the current graphics window.

For example,

```
define lib1 (dbGetLibIdByCellId (geGetEditCell))  
db:1
```

Scheme Extensions for Working With Libraries

The extensions presented in this section are basic to Synopsys tools. With these extensions, you can

- Create libraries, `dbCreateLib` (see page 3-23)
- Specify technology files for libraries, `dbSetTechFileName` (see page 3-52)
- Replace technology files, `dbReplaceTechFile` (see page 3-46)
- Copy libraries, `dbCreateLib` (see page 3-23)

- Rename libraries, dbRenameLib (see page 3-42)
- Close and save libraries, dbCloseLib (see page 3-19)
- Reference other libraries, dbAddRefLib (see page 3-14)
- Replace reference libraries, dbReplaceRefLibByName (see page 3-44)
- Unlink reference libraries, dbDeleteRefLibByName (see page 3-30)
- Attach files to libraries, dbStoreLibAttachedFile (see page 3-53)
- Delete attached files from libraries, dbDeleteLibAttachedFile (see page 3-27)

dbAddLibSchemeFile

Registers a scheme file which contains functions which are available to all parameterized cells used with this library.

Syntax

```
dbAddLibSchemeFile libId "supportFileName"
```

where the arguments are as follows:

libId

Database object ID of the current library from which you want to reference another library.

Valid values:

- Library ID
- (dbGetCurrentLibId)

For more information, see “Specifying Libraries” on page 3-10.

supportFileName

License Required

Milkyway

dbAddRefLib

Creates a reference library. If *dbAddRefLib* successfully links another library to the main library, the Synopsys application displays the database object ID of the reference library.

Prerequisite

The library for which you want to create a reference library must be open in write (W) mode.

Syntax

`dbAddRefLib libId "refLibName"`

where the arguments are as follows:

libId

Database object ID of the current library from which you want to reference another library

Valid values:

- Library ID
- (`dbGetCurrentLibId`)

For more information, see “Specifying Libraries” on page 3-10.

refLibName

Name of library you want the current library to reference

Valid values: Any string with 127 characters or less corresponding to an existing library in the current directory

Related Commands

`dbDeleteRefLibByName` (see page 3-30) deletes a specified reference library.

`dbReplaceRefLibByName` (see page 3-44) replaces a specified reference library with another.

License Required

Milkyway

dbAddIOText

This command adds IO text.

Syntax

```
dbAddIOText cellId padType textType layerNumber textHeight
```

where the arguments are:

cellId The *dbId* for the top level cell.

padType The string, *fakePin* or *pad [pinName]*.

textType Is which data you want to appear in the text,
netName, *portName* or *cellInstName*.

layerNumber Is the integer layer number for the text.

textHeight Is the integer text height. 10 to 30 works well.

Example

```
dbAddIOText (geGetEditCell) "pad" "netName" 5 3
```

License Required

Milkyway

dbAllowToAddAllPGIOText

This function will set the mode for the scheme function dbAddIOText (see page 3-16).

Syntax

```
dbAllowToAddAllPGIOText "booleanValue"
```

where the arguments are as follows:

"booleanValue"

- a scheme boolean

valid values:

#t will cause *dbAddIOText* to add text to all power/ground nets including power/ground nets with multiple pins.

#f will cause *dbAddIOText* to behave as before (default value).

License Required

Milkyway

dbClearCellDifferentialGroup

Clears all the differential group definitions of the design.

Syntax

```
dbClearCellDifferentialGroup cellId
```

where the arguments are as follows:

cellId

cell Id for which nets of the groups belong.

Example

To clear all the differential group definitions of the current design:

```
dbClearCellDifferentialGroup (geGetEditCell)
```

License Required

Milkyway

dbCloseLib

Closes and saves changes to a library. If *dbCloseLib* succeeds in closing and saving your library, the Synopsys application returns *#t*.

Side Effects

All cells within the library are closed.

Syntax

```
dbCloseLib [libId]
```

where *libId* is the database object ID of any library in the current directory that you want to close and save.

Note:

Omission of this argument closes the main library.

Example 1

```
dbOpenLib "train"
db:1
.
.
.
dbCloseLib
#t
```

This example closes the library *train* and all open cells.

Example 2

```
define lib1 (dbOpenLib "train")
db:1
define lib2 (dbOpenLibrary "democ" "r")
db:2
.
.
.
dbCloseLib lib2
#t
```

In this example,

- The first line opens the main library and creates an alias (*lib1*) for the database object ID of the library.
- The second line opens a reference library and creates an alias (*lib2*) for the database object ID of the reference library.
- The third line closes the reference library.

License Required

Milkyway

dbConvertPortToDiodePort

Converts a regular port into a special diode port for axgInsertDiode and axgAddDiodeForPreroute.

Syntax

```
dbConvertPortToDiodePort libId cellName portName
```

where the arguments are as follows:

<i>libId</i>	ID of an opened library: (dbGetCurrentLibId).
<i>cellName</i>	Name of the cell from which the diode port will be converted. Valid values: Name of any cell in the library.
<i>portName</i>	Name of the port to be converted. Valid values: Any port in the cell specified by <i>cellName</i> .

Related Commands

axgAddDiodeForPreroute (see the *IC Implementation Command Reference Manual*).

axgInsertDiode (see the *IC Implementation Command Reference Manual*).

License Required

Milkyway

dbCopyLib

Copies a library. If *dbCopyLib* successfully creates a new library, *toLib*, the Synopsys application returns #*t*.

Prerequisite

The library you want to copy, *fromLib*, must be closed. A Synopsys application supports only one library open for editing.

Syntax

`dbCopyLib "fromLib" "toLib"`

where the arguments are as follows:

fromLib

Name of library you want to copy

Valid values: Name of any library in the current directory

toLib

Name you want to assign the new library

Valid values: Any text string with 255 characters or less

Note:

The Synopsys application does not open the new library, *toLib*, at the end of the copying procedure.

Related Commands

`dbOpenLib` (see page 3-39) opens a library.

License Required

Milkyway

dbCreateLib

Creates a library in the current directory. If *dbCreateLib* successfully creates and opens a new library, the Synopsys application displays the database object ID of the new library.

Syntax

```
dbCreateLib "libName" "techfile.tf" "hierSep"
```

where the arguments are as follows:

libName

Name of library you want to create

Valid values: Any text string 255 characters or less

techfile.tf

Name of an existing technology file. The technology file contains information about the cells in the library.

Note:

You can use any valid technology file to create your library.

After your library is created, you can modify the technology file.

For more information, see “Technology File” on page 1 of the *Milkyway Data Preparation User Guide*.

Valid values: Any text string

hierSep

The character to be used as a separator in hierarchical names. This argument is optional. If you do not enter a character, hierarchical names in the library will be separated by the default backslash (/) character.

Related Commands

dbAddRefLib (see page 3-14) creates a reference library.

dbCloseLib (see page 3-19) closes and saves the library.

dbSetTechFileName (see page 3-52) attaches a technology file to the library.

License Required

Milkyway

dbCreatePcellDefinition

Registers a parameterized cell (Pcell) with the library.

Syntax

```
dbCreatePcellDefinition libId "pcellName" "pcellFileName"
```

where the arguments are as follows:

libId

Database object ID of the current library from which you want to reference another library

Valid values:

- Library ID

- (`dbGetCurrentLibId`)

For more information, see “Specifying Libraries” on page 3-10.

pCellName Defines the name of the Pcell (string).

pCellFileName Defines the full path to the file that contains the parameterized

cell functions.

Note:

Note: At this point, a copy of the *pCellFileName* is stored with the library. If the user modifies the file *pCellFileName*, it will NOT automatically update the Pcell registered in the library.

License Required

Milkyway

dbDefineDifferentialGroup

Defines differential groups. Each group can have two or more nets.

Syntax

```
dbDefineDifferentialGroup cellId groupName netNameList
```

where the arguments are as follows:

cellId

cell Id for which nets of the groups belong.

groupName

Defines the name of the Pcell (string).

netNameList

List of net names you want to form the group.

Example

To define a group “g1” with nets “A” and “B” as its members:

```
dbDefineDifferentialGroup (geGetEditCell) "g1" '( "A" "B" )
```

License Required

Milkyway

dbDeleteLibAttachedFile

Unlinks an attached file from a library. If *dbDeleteLibAttachedFile* successfully unlinks the attached file from the current library, the Synopsys application returns #t.

Caution!

In addition to unlinking the file from your library, *dbDeleteLibAttachedFile* deletes the attached file from the current directory.

Prerequisite

The library from which you want to delete an attached file must be open in write (W) mode.

Syntax

```
dbDeleteLibAttachedFile libId "pseudoName"
```

where the arguments are as follows:

libId

Database object ID of the current library from which you want to delete an attached file

Valid values:

- Library ID
- (dbGetCurrentLibId)

For more information, see “Specifying Libraries” on page 3-10.

pseudoName

User name of attached file you want to delete

Valid values: Name of any file attached to the current library

Related Commands

dbStoreLibAttachedFile (see page 3-53)

License Required

Milkyway

dbDeletePcellDefinition

Removes a parameterized cell from the library.

Syntax

dbDeletePcellDefinition *libId* "*pcellName*"

where the arguments are as follows:

libId

Database object ID of the current library from which you want to reference another library

Valid values:

- Library ID
- (dbGetCurrentLibId)

For more information, see “Specifying Libraries” on page 3-10.

pCellName Defines the name of the Pcell (string).

License Required

Milkyway

dbDeleteRefLibByName

Deletes a reference library from a library. If *dbDeleteRefLibByName* successfully unlinks the reference library from the current library, the Synopsys application returns #t.

Note:

The function dbDeleteRefLibByName does not delete the reference library from your directory.

Prerequisite

The library from which you want to delete a reference library must be open in write (W) mode.

Side Effects

Any cell within the main library that references a cell in the deleted library will be undefined.

Syntax

```
dbDeleteRefLibByName libId "refLibName"
```

where the arguments are as follows:

libId

Database object ID of the current library from which you want to unlink a reference library

Valid values:

- Library ID
- (dbGetCurrentLibId)

For more information, see “Specifying Libraries” on page 3-10.

refLibName

Name of library you want to unlink from the current library

Valid values: Name of any reference library in the specified library

Related Commands

dbReplaceRefLibByName (see page 3-44) replaces the existing reference library.

License Required

Milkyway

dbDowngradeLib

Reverts a library in the current Synopsys tool release format back to the previous release format. If *dbDowngradeLib* successfully reverts your library, the Synopsys application displays *#t*.

Prerequisite

All libraries must be closed.

Syntax

```
dbDowngradeLib "libName"
```

where *libName* is the name of the library you want to revert back to the previous Synopsys tool release format. Valid values are the names of any library in the current directory.

Related Commands

“Working with Cells” on page 3-54 converts the library to the current Synopsys tool release format.

License Required

Milkyway

dbDumpCellIDifferentialGroup

Dumps all the definitions of the differential groups of the design.

Syntax

```
dbDumpCellIDifferentialGroup cellId fileName
```

where the arguments are as follows:

cellId

cell Id for which nets of the groups belong.

fileName

the file name you want dump the differential groups and their nets.

Example

To dump out all the differential groups and their nets to output.file:

```
dbDumpCellDifferentialGroup (geGetEditCell) "output.file"
```

License Required

Milkyway

dbEliminatePins

This command removes all the specified pins from the cell. Use this command before auExtractBlockagePinVia (see Physical Implementation Online Help).

Syntax

```
dbEliminatePins "libName" "cellName" "pinNameFile"
```

where the arguments are as follows:

libName

Name of library.

Valid values: Name of any library in the current directory

cellName

Name of cell from which the pins are deleted.

Valid values: Name of any cell in the current library.

pinNameFile

Name of the file containing all the pin names to be deleted.

Example

```
dbEliminatePins "designLib" "ram32" "pinlist"
```

License Required

Milkyway

dbGetAllLibCellInfo

This command displays useful cell information for all cells in the specified library.

Syntax

```
dbGetAllLibCellInfo "library_name"
```

where *library_name* is the name of an existing library.

License Required

Milkyway

dbGetMaxMetalRoutingLayer

This command allows you to check maximum metal routing layer in metal routing layer.

Note:

The default value of the “maxMetalRoutingLayer” will be set during floorplanning to the highest metal in the techfile.

Use only three or four layers of metal in a five or six layer library.

Syntax

```
dbGetMaxMetalRoutingLayer cellId
```

where the arguments are as follows:

cellId

Database object ID of cell to which net information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

Example

To check the maximum metal routing layer:

```
(dbSetMaxMetalRoutingLayer (geGetEditCell) 4)
(dbGetMaxMetalRoutingLayer (geGetEditCell))
-> should return 4
```

License Required

Milkyway

dbGetOneLibCellInfo

This command displays useful information about the specified cell in the specified library.

Syntax

```
dbGetOneLibCellInfo "library_name" "cell_name"
```

where *library_name* is the name of an existing library and *cell_name* is the name of an existing cell in the library.

License Required

Milkyway

dbListUnitRange

This command displays unit names, parasitic and physical attributes, and the ranges of their values associated with the main library.

Syntax

```
dbListUnitRange "library_name"
```

where *library_name* is the name of the main library.

License Required

Milkyway

dbOpenLib

Opens a library for editing. If *dbOpenLib* successfully opens the library, the Synopsys application returns the database object ID of the library.

Caution!

Do not reopen an already open library using *dbOpenLib*. If you do, you will need to close the library twice.

Prerequisite

No libraries can be open for editing. A Synopsys application supports only one library open for editing.

Syntax

```
dbOpenLib "libName"
```

where *libName* is the name of the library you want to open.

Related Commands

dbCloseLib (see page 3-19) closes an open library.

License Required

Milkyway

dbOpenLibrary

Opens a library in read (R) or write (W) mode. If *dbOpenLibrary* successfully opens the library, the Synopsys application displays the database object ID of the library.

Caution!

Do not use *dbOpenLibrary* to open a library in write (W) mode; instead, use *dbOpenLib* (see page 3-39).

Note:

When opening a library in write (W) mode it is better to use *dbOpenLib*. Unlike *dbOpenLib* (see page 3-39), *dbOpenLibrary* does not specify the database object ID of the library globally. When other commands do not request a library ID, such as *dbOpenCell* (see page 3-76), the following error message may appear:

"ERROR: No library opened."

For more information, see “Specifying Libraries” on page 3-10.

Syntax

`dbOpenLibrary "libName" "mode"`

where the arguments are as follows:

libName

Name of library you want to open

Valid values: Name of any library in the current directory

mode

Indicates whether a library can be edited.

Valid values:

- *r* opens the library specified by *libName* for read-only.
- *w* opens the library specified by *libName* for read and write capabilities.

Related Commands

`dbCloseLib` (see page 3-19) closes an open library.

`dbOpenLib` (see page 3-39) opens a library and specifies the database ID globally. You should use this function when opening a library in write (W) mode.

License Required

Milkyway

dbRenameLib

Renames a library. If `dbRenameLib` succeeds, the Synopsys application returns `#t`.

Prerequisite

The library you want to rename must be closed.

Side Effects

After `dbRenameLib` renames your library, only the renamed library exists. If your renamed library is a reference library, the cells in the main library referencing cells in this library will be undefined.

Replace the reference library of the old name with the renamed library using `dbReplaceRefLibByName`.

Syntax

```
dbRenameLib "oldLibName" "newLibName"
```

where the arguments are as follows:

oldLibName

Name of the existing library to be renamed

Valid values: Name of any library in the current directory

newLibName

Name of the new library to which data from the existing library moves

Valid values: Any text string with less than 255 characters

Note:

The renamed library will not be open after you execute *dbRenameLib*.

Related Commands

dbCreateLib (see page 3-23) copies a library.

dbReplaceRefLibByName (see page 3-44) changes the library referenced by a main library.

License Required

Milkyway

dbReplaceRefLibByName

Replaces an existing reference library in a library. If *dbReplaceLibByName* succeeds, the Synopsys application returns *#t*.

Prerequisite

The library for which you want to replace a reference library must be open in write (W) mode.

Syntax

```
dbReplaceRefLibByName libId "oldRefLibName" "newRefLibName"
```

where the arguments are as follows:

libId

Database object ID of the current library for which you want to replace a reference library

Valid values:

- Library ID
- (dbGetCurrentLibId)

For more information, see “Specifying Libraries” on page 3-10.

oldRefLibName

Name of the existing reference library from which data moves

Valid values: Name of any reference library attached to the current library

newRefLibName

Name of the new reference library to which data from the existing reference library moves

Valid values: Any text string with less than 127 characters

Note:

This command does not open the new reference library.

License Required

Milkyway

Related Commands

`dbAddRefLib` (see page 3-14) creates a reference library for the specified library.

`dbDeleteRefLibByName` (see page 3-30) deletes a reference library from the library.

dbReplaceTechFile

Replaces the existing technology file for a library. If `dbReplaceTechFile` succeeds, the Synopsys application returns `#t`.

Prerequisite

The library for which you want to replace a technology file must be open in write (W) mode.

Syntax

```
dbReplaceTechFile libId "techfile.tfo"
```

where the arguments are as follows:

libId

Database object ID of the current library for which you want to replace the technology file

Valid values:

- Library ID
- (`dbGetCurrentLibId`)

For more information, see “Specifying Libraries” on page 3-10.

techfile.tfo

Name of the technology file that replaces the existing technology file of the current library. This is a binary compiled format of the technology file.

Valid values: Name of any technology file in the current directory

License Required

Milkyway

Related Commands

`dbSetTechFileName` (see page 3-52) attaches a technology file to a library.

dbSetHierRefLibMode

Sets the child cell search order to either the hierarchical reference library mode or to the non-hierarchical reference library mode.

Syntax

`dbSetHierRefLibMode flag`

where `flag` is boolean:

#t - Sets the library cell search order to the hierarchical mode. The current library is the library that contains the parent cell with the cell instance (child cell reference). Cell searching starts in the current library, then traverses through the current library's reference libraries. This is the default setting.

#f - Sets the library search order to the non-hierarchical mode. Cell searching always starts in the top (main) library, then traverses through the top library's reference libraries. The access depth is limited to two levels only.

Example

The following example sets the library search order in hierarchical mode:

```
dbSetHierRefLibMode #t  
#t
```

License Required

Milkyway

dbSetIOPad

This function is used to set the cell flag of all cell instances of the cell (of the specified cellId) to the IO pad cell. However, if the cell flag has been set to standard cell then this command will not change it to IO pad cell.

Syntax

```
dbSetIOPad cellId
```

where:

cellId indicates the internal ID of any open cell in the current library to which you want to set the cell flag.

Related Commands

dbSetStdCell (see page 3-51)

License Required

Milkyway

dbSetMacroCellOrient

Enables you to control the placement orientations of the macro cells. This setting would take effect during the geMove on page 2188 command or the macro placement command.

Note:

The same is true for dbSetStdCellOrient on page 3-284.

Application

This command applies to the applications in the following table.

Application	Tool: <i>Menu Command</i>
Astro	none

Syntax

dbSetMacroCellOrient <libName> <cellName> <orientation>

where the arguments are as follows:

<i>libName</i>	Name of the library containing the cell that the orientation specification applies. Valid values: Name of any library in the current directory.
<i>cellName</i>	Name of the cell that the orientation specification applies. Valid values: <ul style="list-style-type: none">• Name of any cell in the specified library.• Pattern matching to select all cells with names that match a pattern. For example, typing <i>TOP.*</i> specifies all cells with names that begin with <i>TOP</i>. For information about pattern matching, see “Specifying Objects by Pattern” in <i>Physical Implementation Getting Started</i>.
<i>orientation</i>	One of {Any Reflect Noreflect} or: combination of {R0 R0_MX R0_MY R180 R90 R90_MX R90_MY R270} Example: "R0 R0_MY R180 R90_MX" These valid values are case insensitive, and spaced by blanks.

Example

```
dbSetMacroCellOrient "GOLDROAD" "RAM101" "Reflect"
(same as "R0_MX R180")
dbSetMacroCellOrient "GOLDROAD" "ROM102" "R0 R0_MY R180"
```

The first example instructs Astro to place macro instances of RAM101 with either R0_MX or R180 (reflected) orientation. The second example further allows R0 as a legal orientation for any instance of master ROM102.

License Required

Milkyway

dbSetStdCell

This function is used to set the cell flag of all cell instances of the cell (of the specified cellId) to standard cell.

Syntax

```
dbSetStdCell cellId
```

where:

cellId indicates the internal ID of any open cell in the current library to which you want to set the cell flag.

Related Commands

dbSetIOPad (see page 3-48)

License Required

Milkyway

dbSetTechFileName

Attaches a technology file to a library. If *dbSetTechFileName* succeeds, the Synopsys application returns #*t*.

Prerequisite

The library for which you want to attach a technology file must be open in write (W) mode.

Syntax

```
dbSetTechFileName "techFileName"
```

where *techfileName* is the name of the technology file you want to attach to the current library.

License Required

Milkyway

Related Commands

dbReplaceTechFile (see page 3-46) replaces the existing technology file.

dbStoreLibAttachedFile

Attaches a file to a specified library. If *dbStoreAttachedFile* succeeds, the Synopsys application returns #t.

Prerequisite

The library to which you want to attach a file must be open in write (W) mode.

Syntax

```
dbStoreLibAttachedFile libId "attachedFileName"  
"pseudoName"  
"comment"
```

where the arguments are as follows:

libId

Database object ID of the current library to which file attaches

Valid values:

- Library ID
- (dbGetCurrentLibId)

For more information, see “Specifying Libraries” on page 3-10.

attachedFileName

Name of the actual cell to be attached to a library

Valid values: Name of any cell in the current directory

pseudoName

Name that is to be associated with the actual cell

Valid values: Any text string with 16-characters or less

comment

Text that is to be associated with the attached file

Valid values: Any text string with 32-characters or less

License Required

Milkyway

Working with Cells

In the Synopsys environment, a representation of a design is called a cell. For any design, the possible cells include the following:

- A layout cell, which represents the physical layout of a design

- A pin/blockage cell, which represents the pins and blockage areas (areas where no routing on a particular layer can occur)
- A netlist cell, which represents the logical connectivity of a design at the top level of hierarchy
- An expanded netlist cell, which represents the logical connectivity of a design at enough levels of hierarchy to provide information necessary for binding with the physical layout
- A timing cell, which represents the timing-driven constraints for placement, pin information, such as pin access direction
- A notch fill cell
- A gap fill cell
- A hierarchical netlist cell

The table below lists the different types of cells and the naming conventions for the cells and corresponding binary files. Although you can assign any name up to 32 characters to a cell, the extension (after the ".") cannot exceed 4 characters.

Table 4-4

For a cell of type ...	the first version of the cell is named ...	and the corresponding binary file is named ...
layout	<layoutCellName>.CEL;1	CEL/<layoutCellName>:1
pin/blockage	<layoutCellName>.FRAM;1	FRAM/<layoutCellName>:1
netlist	<topNetCellName>.NETL;1	NETL/<topNetCellName>:1
expanded netlist	<topNetCellName>.EXP;1	EXP/<topNetCellName>:1
timing data	<layoutCellName>TIM;1	TIM/<layoutCellName>:1
notch fill	<layoutCellName>.NOTC;1	GAP/<netlistCellName>:1

Table 4-4

For a cell of type ...	the first version of the cell is named ...	and the corresponding binary file is named ...
gap fill	<layoutCellName>.GAP;1	GAP/<layoutCellName>:1
hierarchical netlist	<netlistCellName>.HNET;1	HNET/<netlistCellName>:1

In addition to the design name and extension, cell names include version numbers. The following is the syntax for a complete cell name:

designName : versionNumber

For example, the first version of a cell called *mychip* is *mychip:1*. The next version is *mychip:2*. Sequential version numbers are assigned automatically and cannot be changed.

Guidelines for naming cells include the following:

- The first character should be an alphabetic character.
- The remaining characters may be any of the following:
 - Any alphabetic characters
 - Any numeric characters
 - Underscore (_)
- The length should not exceed 31 characters.

Cell Access

You can open a cell in either of the following modes:

- Read only mode lets you open the cell (read) but not make any changes to it (write).
- Write mode lets you open the cell and make changes (read and write).

When you open a cell, the Synopsys application assigns it a cell ID. (Note that this ID is only valid for the current Synopsys session and while the cell is open.) Most Synopsys Scheme extensions require you to provide this ID as context when modifying cells or working with other database objects. Therefore, always assign this ID to a variable that will remain bound until the cell is closed. For example,

```
(let
  ((cell1 (dbOpenCell "toprt" "w")))
  .
  .
  .
  (dbCloseCell cell1)
)
```

Note:

The variable `cell1` is only available inside the `let` statement. To create a global variable that references the cell ID, use `define`. For information on the use of `define`, see `define variable init` on page 1-17.

In the example, dbOpenCell (see page 3-76) opens *toprt* in write mode. Synopsys applications enforce cell access control as follows:

- When you open a cell in read only mode, the Synopsys application does not place any lock on the cell. Multiple users can open a cell in read only mode.
- When you open a cell in write mode, the Synopsys application locks the cell from write access by other users by creating a lock file (*<cellName>.lock*). When you close the cell, the Synopsys application removes the lock file, thus allowing write access to the cell by another user.

If you open a cell in read only mode and want to make changes (write), you can change the access mode as follows:

```
dbChangeAccessMode cellId "w"
```

where *cellId* is the internal ID of any open cell in the current library to which you want to make changes. The argument *cellId* can be either of the following:

- Cell ID
- (geGetEditCell)

For more information, see “Specifying Cells” on page 3-57.

Specifying Cells

Most of the Synopsys system extensions require you to specify a cell ID. You can do this two ways:

- If you opened the cell using dbOpenCell (see page 3-76), you must specify a variable for *cellId*.

To specify a cell ID, bind the ID to a variable when you open a cell and use the variable for *cellId*. For example,

```
define cell1 (dbOpenCell "toprt" "w")
db:2
.
.
.
dbCheckCell cell1
bbox on pip (2147483647, 2147483647)
(-2147483648, -2147483648)
bbox computed (0, 0) (12000, 52500)
.
.
.
#t
```

- If you opened the cell using the *Cell > Open* menu command, *geOpenCell* (see page 2233 of the *IC Implementation Command Reference Manual*), you can substitute the following for *cellId*:

```
(geGetEditCell)
```

For more information, see *geGetEditCell* on page 4-4.

In the previous example, you substitute (*geGetEditCell*) for *cell1* as follows,

```
dbCheckCell (geGetEditCell)
bbox on pip (2147483647, 2147483647)
(-2147483648, -2147483648)
bbox computed (0, 0) (12000, 52500)
.
.
.
#t
```

If the cell ID is not bound to a variable or the variable is lost, you can do one of the following:

- If you opened the cell using `dbOpenCell` (see page 3-76), reopen the cell using `dbOpenCell` in the following syntax:

```
define identifier (dbOpenCell "cellName" "w")
```

where the arguments are as follows:

identifier

Name of the variable to which the cell ID binds

Valid values: Any text string

cellName

Name of cell to which information applies

Valid values: Name of any cell in the current library

Note:

Binding the cell ID to a variable does not erase or save any changes you have made.

- If you opened the cell with the *Cell > Open* menu command, type the following:

```
define cell1 (geGetEditCell)
```

Scheme Extensions for Working with Cells

The extensions presented in this section are basic to Synopsys tools. With these extensions, you can:

- Create cells, `dbDeleteCell` (see page 3-68)
- Copy cells, `dbCopyCell` (see page 3-63)
- Rename cells, `dbRenameCell` (see page 3-77)
- Close and save cells, `dbCloseCell` (see page 3-62)
- Close without saving, `dbAbortCell` (see page 3-60)
- Save cells while open, `dbSaveCell` (see page 3-80)
- Revert cell to last saved cell version, or lose current edits, and open, `dbRollBackCell` (see page 3-78)
- Attach files to a cell, `dbStoreCellAttachedFile` (see page 3-81)
- Delete attached files from a cell, `dbSaveCell` (see page 3-80)

dbAbortCell

Closes a cell without saving your changes. If `dbAbortCell` succeeds, the Synopsys application returns `#t`.

Syntax

`dbAbortCell cellId`

where *cellId* is the database object ID of any open cell in the current library that you want to close without saving. The argument *cellId* can be either of the following:

- Cell ID
- (geGetEditCell)

For information, see “Specifying Cells” in *Physical Implementation Getting Started*.

License Required

Milkyway

Related Commands

[dbCloseCell](#)—closes the cell and saves your changes.

[dbRollBackCell](#)—deletes your changes and keeps the cell open.

[dbSaveCell](#)—saves your changes and keeps the cell open.

dbChangeAccessMode

Changes the current cell access mode from read-only (R) to write (W). If *dbChangeAccessMode* succeeds, the Synopsys application returns #t.

Syntax

```
dbChangeAccessMode cellId "accessMode"
```

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” in *Physical Implementation Getting Started*.

accessMode

Indicates whether you can edit the cell.

Valid value: *w* means that edits are allowed.

License Required

Milkyway

dbCloseCell

Closes the cell and saves your changes. If *dbCloseCell* succeeds, the Synopsys application returns *#t*.

Syntax

`dbCloseCell cellId`

where *cellId* is the database object ID of any open cell in the current library that you want to close and save. The argument *cellId* can be either of the following:

- Cell ID
- (geGetEditCell)

For information, see “Specifying Cells” in *Physical Implementation Getting Started*.

License Required

Milkyway

Related Commands

dbAbortCell (see page 3-60) closes an open cell without saving your changes.

dbRollBackCell (see page 3-78) deletes your changes and keeps the cell open.

dbSaveCell (see page 3-80) saves your changes and keeps the cell open.

dbCopyCell

Creates a duplicate copy of a cell. If successful, *dbCopyCell* returns the database ID of *toCell*. If not successful, it returns a pair of parentheses (). Obviously the Scheme version of copy cell copies *only* the specified cell. The scheme application program should copy any additional cell views desired.

Note:

Both fromCell and toCell are open after execution of dbCopyCell.

Caution!

Although you can execute dbCopyCell without binding the cell Id of toCell to a variable, it is not recommended. Most scheme commands refer to cells by their cell ID. If the cell ID has not been bound to a variable, some commands cannot modify the cell. For example, cellID is one of the arguments of dbSaveCell. For more information, see “Specifying Cells” on page 3-57.

Syntax

dbCopyCell "fromCell" "toCell"

where the arguments are as follows:

fromCell

Name of cell you want to copy

Valid values: Name of any cell in the current library

toCell

Name of duplicate cell you want to create

Valid values: Any string with 31 characters or less

License Required

Milkyway

Related Commands

dbCopyCellToAnotherLibDisk (see page 3-67)

dbCopyCellFromAnotherLib (see page 3-65)

dbCopyCellFromAnotherLib

Copies a cell from another library to the current library. If *dbCopyCellFromAnotherLib* succeeds, the Synopsys application returns the database object ID of the copied cell.

Note:

The copied cell, *toCell*, is open after execution of *dbCopyCellFromAnotherLib*.

Caution!

Although you can execute *dbCopyCellFromAnotherLib* without binding the cell Id of *toCell* to a variable, it is not recommended. Most scheme commands refer to cells by their cell ID. If the cell ID has not been bound to a variable, some commands cannot modify the cell. For example, *cellID* is one of the arguments of *dbSaveCell*. For more information, see “Specifying Cells” on page 3-57.

Prerequisite

The library, *fromLib*, where the cell to be copied is located, must be closed. A Synopsys application supports only one library open at a time for editing.

Syntax

```
dbCopyCellFromAnotherLib "fromCell" "fromLib" "toCell"
```

where the arguments are as follows:

fromCell

Name of cell you want to copy

Valid values: Name of any cell in the current directory

fromLib

Name of library in which the cell you want to copy is located

Valid values: Name of any library in the current directory containing *fromCell*

toCell

Name of cell to you want to create

Valid values: Any string with 31 characters or less

License Required

Milkyway

Related Commands

dbCopyCell (see page 3-63)

dbCopyCellToAnotherLibDisk (see page 3-67)

dbCopyCellToAnotherLibDisk

Copies a cell to another library from the current library. If successful, dbCopyCell (see page 3-63) returns #t.

Prerequisite

The library in which you want to place the copied cell needs to be closed.

Syntax

```
dbCopyCellToAnotherLibDisk "fromCell" "toLib" "toCell"
```

where the arguments are as follows:

fromCell

Name of cell you want to copy

Valid values: Name of any cell in the current library

toLib

Name of library to which you want to copy *fromCell*

Valid values: Name of any library in the current directory

toCell

Name of cell you want to create

Note:

If a cell named *toCell* already exists in *toLib*, the Synopsys application creates a new version of the cell. For example, the application creates *toCell.extension;versionNumber+1*.

Valid values: Any string with 31 characters or less

License Required

Milkyway

Related Commands

dbCopyCell (see page 3-63)

dbCopyCellFromAnotherLib (see page 3-65)

dbDumpCell (see page 3-71)

dbDeleteCell

Deletes a cell. If *dbDeleteCell* succeeds, the Synopsys application returns *#t*.

Prerequisite

The cell you want to delete must be closed. In addition, if you want to delete a cell from a reference library, the reference library must be open in write (W) mode.

Syntax

```
dbDeleteCell "cellName"
```

where the *cellName* is the name of the cell you want to delete. Valid values are the names of any cell in the current library.

License Required

Milkyway

dbDeleteCellAttachedFile

Deletes an attached file from a cell. If *dbDeleteCellAttachedFile* succeeds, the Synopsys application returns *#t*.

Prerequisite

The cell from which you want to delete an attached file must be open in write (W) mode.

Syntax

```
dbDeleteCellAttachedFile cellId "pseudoName"
```

where the arguments are as follows:

cellId

Database object ID of the cell from which you want to delete an attached file

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” in *Physical Implementation Getting Started*.

pseudoName

User name of attached file you want to delete

Valid values: Name of any attached file in the specified library

License Required

Milkyway

dbDeleteInstAndStitch

Deletes all cell instances in the specified file. *dbDeleteInstAndStitch* handles only one-input and one-output, then stitches all its fanouts to the input net.

Syntax

```
dbDeleteInstAndStitch cellId fileName
```

where the arguments are as follows:

cellId

Database object ID of the cell from where you want to delete all cell instances in the specified file.

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” in *Physical Implementation Getting Started*.

fileName

Name of the file where you want to delete all cell instances.

Valid values: Any string up to 32 characters

License Required

Milkyway

dbDumpCell

Dumps the cell content to a file.

Syntax

```
dbDumpCell cellId "fileName"
```

where the arguments are as follows:

cellId

Database object ID of the cell from which you want information

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” in *Physical Implementation Getting Started*.

fileName

Name of file to which cell content dumps

Valid values: Any text string

License Required

Milkyway

dbDumpFreezeRoutingNets

This feature will dump a list of frozen nets to the file. The dumped file can be modified manually and loaded back again.

Syntax

```
dbDumpFreezeRoutingNets cellId option "dumpFileName"
```

where the arguments are as follows:

cellId

Database object ID of the cell from which you want to delete an attached file

Valid values:

- Cell ID of any open cell in the current library.
- (geGetEditCell).

For information, see “Specifying Cells” in *Physical Implementation Getting Started*.

option

Boolean where:

#t creates a dump list of frozen nets to a file.

#f creates a dump list of nets that can be changed freely.

dumpFileName

Name of file to which cell content dumps.

Valid values: Any text string.

Examples

The following example dumps a list of frozen nets to the file, ‘freeze.list.’:

```
dbDumpFreezeRoutingNets (geGetEditCell) #t "freeze.list"
```

The following example dumps a list of nets that are not frozen to ‘routable.list.’

```
dbDumpFreezeRoutingNets (geGetEditCell) #f "routable.list"
```

License Required

Milkyway

dbFreezeRoutingOnNets

This feature will prevent the router from changing the specified nets. The status will be stored in the database, and will constrain all of the following routing operations.

Syntax

```
dbFreezeRoutingOnNets cellId option "net_name_pattern"
```

where the arguments are as follows:

cellId

Database object ID of the cell from which you want to delete an attached file

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” in *Physical Implementation Getting Started*.

option

Boolean where

#t freezes the routing matching the pattern specified in *net_name_pattern*

“m” allows minor changes on the net

#f allows the router to change nets matching the pattern specified in the *net_name_pattern*

net_name_pattern

Name of specified nets for which routing is to be frozen or which the router is to change

Valid values: Any text string

Example

The following example freezes routing on nets matching the pattern, a/ .* or b/ .*

```
dbFreezeRoutingOnNets (geGetEditCell) #t "a/.*, b/.*"
```

The following example allows the router to change nets matching the pattern, a/ .*

```
dbFreezeRoutingOnNets (geGetEditCell) #f "a/.*"
```

License Required

Milkyway

dbOpenCell

Opens the specified cell in read (R) or write (W) mode. If *dbOpenCell* succeeds, the Synopsys application returns the database object ID of the cell and opens the cell in the specified mode.

Prerequisite

The library containing the cell you want to open must be open.

Syntax

```
dbOpenCell "cellName" "mode"
```

where the arguments are as follows:

cellName

Name of cell to open

Valid values: Name of any cell in the current library

mode

Indicates whether you can edit the cell.

Valid values:

- *r* opens the cell for reading only.
- *w* opens the cell for read and write capabilities.

License Required

Milkyway

Related Commands

dbAbortCell (see page 3-60) closes an open cell without saving your changes.

`dbChangeAccessMode` (see page 3-61) changes an cell opened in read (R) mode to write (W) mode.

`dbCloseCell` (see page 3-62) closes an open cell.

`dbRenameCell` (see page 3-77) changes the name of your cell.

dbRenameCell

Renames a cell. If `dbRenameCell` succeeds, the Synopsys application returns `#t`.

Note:

This function does not create a copy of the specified cell. After execution of `dbRenameCell`, the original cell no longer exists.

Prerequisite

The cell you want to reload must be closed.

Syntax

```
dbRenameCell "oldCellName" "newCellName"
```

where the arguments are as follows:

oldCellName

Name of the existing cell you want to rename

Valid values: Name of any cell in the current library

newCellName

Name you want to assign to the cell

Valid values: Any text string with 31 characters or less

License Required

Milkyway

Related Commands

dbCopyCell (see page 3-63) creates a duplicate of your cell.

dbCopyCellToAnotherLibDisk (see page 3-67) creates and places a duplicate of your cell in another library.

dbRollBackCell

Replaces the current cell with the last version of the cell saved to the disk. This discards any changes made since you last saved the cell. Unlike dbAbortCell (see page 3-60), *dbRollBackCell* opens the last version of the cell that was saved to the disk.

If *dbRollBackCell* succeeds, the Synopsys application returns #t and opens the last version of the cell that was saved to disk.

Note:

The function dbRollBackCell does not replace the current cell in the graphics window. Use Views > Redraw to update the graphics window.

Syntax

`dbRollBackCell cellId`

where the arguments are as follows:

cellId

Database object ID of the cell you want to rollback

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” in *Physical Implementation Getting Started*.

License Required

Milkyway

Related Commands

dbAbortCell (see page 3-60) closes an open cell without saving your changes.

dbCloseCell (see page 3-62) closes an open cell and saves your changes.

dbSaveCell (see page 3-80) saves your changes and keeps the cell open.

dbSaveCell

Saves the cell in the current window to disk. If *dbSaveCell* succeeds, the Synopsys application returns #.

Syntax

`dbSaveCell cellId`

where the arguments are as follows:

cellId

ID of the cell to save

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” in *Physical Implementation Getting Started*.

License Required

Milkyway

Related Commands

dbAbortCell (see page 3-60) closes an open cell without saving your changes.

dbCloseCell (see page 3-62) closes an open cell and saves your changes.

dbRollBackCell (see page 3-78) deletes your changes and keeps the cell open.

dbStoreCellAttachedFile

Attaches a file to a cell. If *dbStoreCellAttachedFile* succeeds, the Synopsys application returns #*t*.

Syntax

```
dbStoreCellAttachedFile cellId "attachedFileName"  
"pseudoName" "comment"
```

where the arguments are as follows:

cellId

Database object ID of the cell to which you want to attach a file

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” in *Physical Implementation Getting Started*.

attachedFileName

Actual name of the file to be attached to a cell

Valid values: Name of any file in the current library

pseudoName

User name you want to associate with the attached file

Valid values: Any text string with 16-characters or less

comment

Text that is to be associated with the attached file

Valid values: Any text string with 32-characters or less

License Required

Milkyway

Extensions for Querying the Database

This section provides syntax information for extensions that provide the ability to query the Synopsys databases.

Note:

Unless otherwise specified, the examples are from the Astro product.

->?

Lists the valid field names for a cell or library. This function is used for program development.

Syntax

`->? id`

where *id* is the database object ID of any cell or library in the current directory from which you want a list of valid field names.

Example - List valid field names for a library.

```
define lib1 (dbOpenLib "ex4")
db:1
.
.
.
->? lib1
"attachedFile, catalog, libHeader, object, eqClass,
property, cellRecordNumber"
```

In this example, the function `->?` lists the fields of the library *lib1*.

Note:

The function dbFetchObject (see page 3-95) is more versatile than ->?. Unlike ->?, dbFetchObject is not limited to cells and libraries; it lists both the field names and values for any database object.

License Required

Milkyway

Related Commands

dbFetchObject (see page 3-95)

->type

Identifies whether a database ID is a library ID or not.

Syntax

`->type objectId`

where *objectId* is a database object ID of any object in the current directory from which you want ID information.

If *objectId* is a library ID, ->type returns "DB library Id". If *objectId* is not a library ID, ->type returns the empty list, () .

Example - Verify whether a database ID is a library ID or not.

```
define lib1 (dbOpenLib "design")
db:1
define cell1 (dbOpenCell "top" "w")
db:2
.
.
.
->type lib1
"DB library Id"
->type cell1
()
```

In this example, `->type` confirms that the variable `lib1` is a library ID and `cell1` is not.

License Required

Milkyway

dbCloseQueryFile

Closes the current query file. If `dbCloseQueryFile` is successful, the Synopsys application returns `#t`.

Syntax

`dbCloseQueryFile`

License Required

Milkyway

clfDump

Writes the CLF information associated with a library to a file.

Syntax

```
clfDump libID fileName
```

where the arguments are as follows:

libID

ID of the library for which you want CLF information

Valid values: ID of any library in the current directory. Either of the following can be substituted for *libID*:

- (dbGetCurrentLibId)
- If a cell is open in your library,
(dbGetLibIdByCellId (*geGetEditCell*))

fileName

Name of the file to which you want to write the information

Valid values: Any string up to 32 characters

Example - Dump the CLF information for the library *demo*.

```
clfDump (dbOpenLib "demo") "demo.clf"
```

Typing this function from the input area of the Milkyway window writes the CLF information associated with the library *demo* to a file named *demo.clf*.

License Required

Milkyway

dbFetchContactGeom

Returns the index number for a layer. With this function, you can convert a contact into real physical objects.

Syntax

```
dbFetchContactGeom cellId objectID
```

where the arguments are as follows:

cellId

Database object ID of any open cell in the current library from which you want information about contact geometry

Valid values:

- Cell ID
- (geGetEditCell)

For more information, see “Specifying Cells” on page 3-57.

objectID

Database object ID of either the contact or contact array for which you want information about contact geometry

Valid values:

- Contact ID
- Contact array ID

Note:

Using a database object ID other than those stated above is an error, and causes *dbFetchContactGeom* to return *nil* or *(())*.

License Required

Milkyway

dbFetchLayerIdByName

Returns the index number for a layer. This function returns *nil* or `()` if the layer specified in *layerName* does not exist.

Syntax

```
dbFetchLayerIdByName libId "layerName"
```

where the arguments are as follows:

libId

Database object ID of the library currently open in write (W) mode from which you want information about a layer

Valid values:

- Library ID of the current library
- `(dbGetCurrentLibId)`

For more information, see “Specifying Libraries” on page 3-10.

layerName

Name of the layer for which you want the index number

Valid values: Name of any layer in the current library

License Required

Milkyway

dbFetchLayerIdByNumber

Returns the index number for a layer. This function returns *nil* or `()` if the layer specified in *layerNumber* does not exist.

Syntax

`dbFetchLayerIdByNumber libId layerNumber`

where the arguments are as follows:

libId

Database object ID of the library currently open in write (W) mode from which you want information about a layer

Valid values:

- Library ID of the current library
- `(dbGetCurrentLibId)`

For more information, see “Specifying Libraries” on page 3-10.

layerNumber

Number of the layer for which you want the index number

Valid values: Positive integer corresponding to any layer in the current library

License Required

Milkyway

Related Commands

`dbFetchLayerIdByName` (see page 3-89)

dbFetchLayerIdList

Returns a list of the index numbers for all layers in a library.

Syntax

```
dbFetchLayerIdList libId
```

where *libId* is the database object ID of the library currently open in write (W) mode from which you want information about a layer. The argument *libId* can be one of the following:

- Library ID of the current library
- (dbGetCurrentLibId)

For more information, see “Specifying Libraries” on page 3-10.

License Required

Milkyway

dbFetchLayerInfo

Returns the technology information for a layer in the *layerTable* section of a technology file. This function returns current value for *layerTechType*. If *layerTechType* does not exist or *libId* is a bad database object ID, *dbFetchLayerInfo* returns *nil* or *(())*.

Syntax

```
dbFetchLayerInfo libId layerIndex "layerTechType"
```

where the arguments are as follows:

libId

Database object ID of the library currently open in write (W) mode from which you want information about a layer

Valid values:

- Library ID of the current library
- (dbGetCurrentLibId)

For more information, see “Specifying Libraries” on page 3-10.

layerIndex

Index that identifies the layer

Valid values: Execute either of the following functions:

- (dbFetchLayerIdByName *libId "layerName"*)
- (dbFetchLayerIdByNumber *libId layerNumber*)

layerTechType

Layer technology type, such as display attributes, layout attributes, for which you want the current value

Valid values: Execute dbFetchLayerTypes on page 3-94 for a list of valid values.

License Required

Milkyway

Related Commands

dbFetchLayerIdByName (see page 3-89)

`dbFetchLayerIdByNumber` (see page 3-90)

`dbFetchLayerTypes` (see page 3-94)

dbFetchLayerTypes

Lists all valid layer technology types.

Syntax

`dbFetchLayerTypes`

License Required

Milkyway

Related Commands

`dbFetchLayerInfo` (see page 3-92)

dbFetchObject

Lists all the field names and the data that is contained within these fields for a given object. This function is a useful programming tool during the development and debugging of Scheme programs. The function `dbFetchObject` returns the field names and current values in the following syntax:

({*fieldName* (*fieldValue*) } ...)

Syntax

`dbFetchObject libraryOrCellId subObjectId`

where the arguments are as follows:

libraryOrCellId

Database object ID of the cell or library for which you want field names and current values

Valid values:

- Cell ID of any cell in the current library
- (geGetEditCell)
- Library ID of the current library
- (dbGetCurrentLibId)

For more information, see “Specifying Libraries” on page 3-10.

subObjectId

Database object ID of the object for which you want field names and current values

Valid values:

- *nil* returns the information contained in the cell header record if *libraryOrCellId* is a cell ID or returns the information contained in the library header record if *libraryOrCellId* is a library ID.
- Object ID of any object within the library or cell specified in *libraryOrCellId* returns all available information for the object.

Example - List all available information for a specified path.

```
dbFetchObject cellId pathId
  ↴( (layerNumber (10)) (pathWidth (5000)) ... )
```

Note:

Both cellId and pathId are defined as variables in a previous step.

License Required

Milkyway

Related Commands

->? (see page 3-83)

dbFetchObjectField (see page 3-97)

dbFetchObjectField

Retrieves the value of a specified field associated with a specified database object. This function is central to any Scheme program that accesses the Synopsys database.

Syntax

dbFetchObjectField fileId objectId fieldName

where the arguments are:

fileId

Database object ID of the cell or library that you want to fetch the value of a specific field.

Valid values:

File ID of any cell in the current library (*geGetEditCell*). For more information, see “Specifying Cells” on page 3-57.

<i>objectId</i>	Database object ID of the object that field information applies.
	Valid values: Object ID of any object within the cell. Objects do not have to be homogeneous within the group. For example, you could place a net and a cell together in one group.
<i>fieldName</i>	Name of the field that information is desired. Valid values: Name of any field associated with the specific object. If the first character of <i>fieldName</i> is '?', the command will display all valid field names associated with <i>subObjectId</i> and/or <i>libraryOrCellId</i> .

Example 1

You can select a wire in the graphics window and query the selected wire and you will see that wire ID is one of the printed parameters. Alternately, from a script *wireId* can be set in a variable from a single selected wire with the command:

```
(wireId (car (geGetSelectedSet (hiGetCurrentWindow))))
(wireMasterNumber (dbFetchObjectField (geGetEditCell)
  wireType
  "wireMasterIdx"))
(wireMasterId (dbMakeWireMasterId (geGetEditCell)
  wireType))
dbFetchObjectField (geGetEditCell) wireType "width"
```

returns the width of the wire. Note that this wire width must be divided by the value of “length precision” in the technology file to obtain the actual wire width. The unit value of the resulting wire width is in units of “*unitLengthName*” from the technology file.

Example 2

```
dbFetchObjectField (geGetEditCell) InstanceId "bbox"  
((252.8 406.2) (288 451))
```

The record name “bbox” is not printed for the *InstanceId* record when queried since it is computed at run time and is not stored in the database. It is, nevertheless, a valid query parameter for the *InstanceId* record.

Example 3

```
define libId (dbOpenLib "DESIGN")  
dbFetchObjectField libId '()' "?"  
    schemaMajorVersion  
    schemaMinorVersion  
    libFlag  
    userDistUnit  
    DataBaseUnitPerUserDistUnit  
    DataBaseUnitPerMinGrid  
    libCreator  
    techFileName  
    libraryName
```

Example 4

```
define cellId (geGetEditCell)  
define portId (dbGetPortByName cellId "A")  
dbFetchObjectField cellId portId "?"  
    name  
    index  
    direction  
    type  
    numberofPin  
    netId  
    propertyId  
    mustJoinRootId
```

Note that “escaped” port names must be handled as an exception with Milkyway scheme since the character is used to denote the character immediately following it in a string is to be interpreted as a literal. So to use `dbGetPortByName` with an escaped port name such as `\R[11]`, the command would be:

```
dbGetPortByName cellId "\\"R[11]"
```

Note that when using the command `geNameSelect` you can simply enter the escaped port name as `\R[11]` and the command will automatically convert the name to `\\"R[11]`.

Related Commands

->? (see page 3-83)

`dbFetchObject` (see page 3-95)

License Required

Milkyway

dbFetchPinInstByPortInst

Returns the pin instance data associated with a port instance.

Syntax

```
dbFetchPinInstByPortInst cellId portInstId
```

where the arguments are as follows:

cellId

Database object ID of cell containing the port instance from which you want pin instance information

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

portInstId

Database object ID of the port instance from which you want pin instance information

Valid values: ID of any port instance in the specified cell

License Required

Milkyway

dbFetchScanTypes

Lists the available scan types associated with the specified library, cell, or sub-object. This function returns the available scan types in the following syntax:

```
( { "scantype" } . . . )
```

Syntax

```
dbFetchScanTypes libraryOrCellId subObjectIdOrNil
```

where the arguments are as follows:

libraryOrCellId

Database object ID of the cell or library for which you want the available scan types

Valid values:

- Cell ID of any cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

- Library ID of the current library
- (dbGetCurrentLibId)

For more information, see “Specifying Libraries” on page 3-10.

subObjectIdOrNil

Database object ID of the object from which information is desired or *nil*

Valid values:

- '() returns the scan types associated with the library or cell.
- Object ID of any object within the specified library or cell returns the scan types associated with the object.

Example - List available scan types for a specified net.

```
dbFetchScanTypes cellId netId  
  ("Property" "ChildNet" "NetMember")
```

Note:

Both `cellId` and `netId` are defined as variables in a previous step.

License Required

Milkyway

dbFetchTechInfo

Returns the value of a technology type. This function returns current value for `techType`. If `techType` does not exist or `libId` is a bad database object ID, `dbFetchTechInfo` returns `nil` or `(())`.

Syntax

```
dbFetchTechInfo libId "techType"
```

where the arguments are as follows:

libId

Database object ID of the library currently open in write (W) mode from which you want information about the technology file

Valid values:

- Library ID of the current library
- `(dbGetCurrentLibId)`

For more information, see “Specifying Libraries” on page 3-10.

techType

Technology type for which you want the current value

Valid values: Execute dbFetchTechTypes (see page 3-104) for a list of valid values.

License Required

Milkyway

Related Commands

dbFetchTechTypes (see page 3-104)

dbFetchTechTypes

Lists all valid technology types.

Syntax

dbFetchTechTypes

License Required

Milkyway

Related Commands

dbFetchTechInfo (see page 3-103)

dbGetAllInstFromFence

Returns a list of all cell instances associated with a fence.

Syntax

dbGetAllInstFromFence *cellId* *fenceId*

where the arguments are as follows:

cellId

Database object ID of cell containing the fence from which you want cell instance information

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

fenceId

Database object ID of the fence from which you want cell instance information

Valid values: Object ID of any fence in the specified cell

License Required

Milkyway

dbGetAllInstFromGroup

Returns a list of all cell instances associated with a group.

Syntax

`dbGetAllInstFromGroup cellId groupId`

where the arguments are as follows:

cellId

Database object ID of the cell containing the group from which you want cell instance information

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

groupId

Database object ID of the group from which you want cell instance information

Valid values: Object ID of any group in the specified cell

License Required

Milkyway

dbGetCellInstByName

Returns the database object ID of a cell instance. If the cell instance does not exist, the function returns *nil*.

Syntax

```
dbGetCellInstByName cellId "cellInstName"
```

where the arguments are as follows:

cellId

Database object ID of the cell containing the cell instance.

Valid values:

- Cell ID of any cell in the current library.
- (geGetEditCell).

For information, see “Specifying Cells” on page 3-57.

cellInstName

Name of a cell instance to which information applies.

Valid values: Local name of any cell instance in the specified cell.

Example

Get the database object ID for a specified cell instance given a local instance name.

```
define cellId (geGetEditCell)
ždb:2
define instanceId (dbGetCellInstByName cellId "_AAAB.I20")
ždb:3001f
```

License Required

Milkyway

Related Commands

dbConvertRegionToMacroCell (see page 3-138)

clfGetCLFLibName

When called within the context of a clf load, returns the library name that the clf load is modifying.

Syntax

`clfGetCLFLibName`

License Required

Milkyway

clfGetCLFLibId

When called within the context of a clf load, returns the library ID that the clf load is modifying.

Syntax

`clfGetCLFLibId`

License Required

Milkyway

dbGetConstraintsCellId

Returns the cell ID of a cell currently operated on by a load constraint.

Syntax

`dbGetConstraintsCellId`

License Required

Milkyway

dbGetCurrentLibId

Returns the library ID of the library currently open in write (W) mode. This function returns the library ID of the library currently open in write (W) mode. If no library is open, *dbGetCurrentLibId* returns *nil* or *(())*.

Syntax

`dbGetCurrentLibId`

License Required

Milkyway

Related Commands

“Specifying Libraries” on page 3-10

dbGetLayerNameFromNumber

Returns the layer name of a specified layer.

Syntax

`dbGetLayerNameFromNumber libId layerNumber`

where the arguments are as follows:

libId

Database object ID of the library to which layer information applies

Valid values: Library ID of any library in the current directory

layerNumber

Number of the layer of which you want the name

Valid values: Number of any layer in the specified library

Example -Determine the name of a given layer number.

```
define libId (dbOpenLib "test")
ždb:1
dbGetLayerNameFromNumber libId 9
ž "METAL1"
```

License Required

Milkyway

Related Commands

dbGetLayerNumberFromName (see page 3-114)

dbGetLayerNumberFromName

Returns the layer number of a specified layer.

Note:

When the cellId is given for the

Syntax

```
dbGetLayerNumberFromName libId "layerName"
```

where the arguments are as follows:

libId

Database object ID of the library to which layer information applies

Valid values: Library ID of any library in the current directory

Note:

When the cellId is used as the libId, the correct layer number is returned.

layerName

Name of the layer of which you want the layer number

Valid values: Name of any layer in the specified library

Example -Determine which layer number corresponds to *metal1*.

```
define libId (dbOpenLib "test")
 db:1
dbGetLayerNumberFromName libId "METAL1"
 9
```

License Required

Milkyway

Related Commands

dbGetLibIdByCellId (see page 3-116)

dbGetLibIdByCellId

Returns the library ID of the library which contains a specified cell.

Syntax

```
dbGetLibIdByCellId cellId
```

where *cellId* is the internal ID of any open cell in the library for which you want the library ID. Valid values can be one of the following:

- Cell ID
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

License Required

Milkyway

Related Commands

dbGetCurrentLibId (see page 3-112)

dbGetNetByName

Returns the database object ID of a net. If the net does not exist, the function returns *nil*.

Syntax

```
dbGetNetByName cellId "netName"
```

where the arguments are as follows:

cellId

Database object ID of the cell containing the net of which you want the database object ID

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

netName

Name of net of which object ID desired

Valid values: Local name of any net in the specified cell

**Example 1 -Get the database object ID for the power net.
For cells opened using dbOpenCell (see page 3-76):**

```
define cellId (dbOpenCell "top" "r")
db:2
.
.
.
dbGetNetByName cellId "VDD"
db:12b002b
```

For cells open in the graphics window:

This example demonstrates the substitution of *geGetEditCell* for *cellId*.

```
dbGetNetByName (geGetEditCell) "VDD"
db:12b002b
```

Both variations of example 1 return the net ID of power net, VDD.

Example 2 -

Test for the existence of net VDD, and create it if it does not exist.

```
(if (not (dbGetNetByName cellId "VDD"))
  (dbCreateNet cellId "VDD")
)
db:12b002b
dbSetNetTypeByName cellId "VDD" "power"
#t
```

License Required

Milkyway

Related Commands

dbCreateNet (see page 3-167) for another example of
dbGetNetByName

dbMoveAllNetMember (see page 3-225) for another example of
dbGetNetByName

dbGetPinByName

Returns the database object ID of a pin. If the pin does not exist, the function returns *nil*.

Syntax

```
dbGetPinByName cellId "pinName"
```

where the arguments are as follows:

cellId

Database object ID of the cell containing the pin of which you want the database object ID

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

pinName

Name of pin of which object ID desired

Valid values: Name of any pin in the specified cell

License Required

Milkyway

dbGetPortByName

Returns the database object ID of a port. If the port does not exist, the function returns *nil*.

Syntax

`dbGetPortByName cellId "portName"`

where the arguments are as follows:

cellId

Database object ID of the cell containing the port of which you want the database object ID

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

pinName

Name of port of which object ID desired

Valid values: Name of any port in the specified cell

License Required

Milkyway

dbGetRegionIdByCellInstId

Gets the region ID from a specified cell instance.

Syntax

```
dbGetRegionIdByCellInstId cellId cellInstId
```

where the arguments are as follows:

cellId

Database object ID of the cell containing the region of which you want the database object ID

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

cellInstId

Database object ID of the cell to which region information applies

Valid values: Cell instance ID of any cell instance in the specified cell

License Required

Milkyway

dbGetRegionPortByName

Gets the region port ID of a specified region port.

Syntax

```
dbGetRegionPortByName cellId "regionPortName"
```

where the arguments are as follows:

cellId

Database object ID of the cell containing the region port of which you want the database object ID.

Valid values:

- Cell ID of any open cell in the current library.
- (geGetEditCell).

For information, see “Specifying Cells” on page 3-57.

regionPortName

Name of region port of which object ID desired.

Valid values: Name of any region port in the specified cell.

License Required

Milkyway

dbGetRegionPortDirection

Gets the direction of a specified region port.

Syntax

```
dbGetRegionPortDirection cellId regionPortId
```

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies.

Valid values:

- Cell ID of any open cell in the current library.
- (geGetEditCell).

For information, see “Specifying Cells” on page 3-57.

regionPortId

Database object ID of the region port from which direction information desired.

Valid values: Region port ID of any region port in the specified cell.

License Required

Milkyway

dbId?

Identifies whether or not a variable is a database object ID.

Syntax

`dbId? variable`

where *variable* can be any variable for which you want to confirm whether or not it is a database object ID.

License Required

Milkyway

Related Commands

`->type` (see page 3-85) identifies if a database object ID is a library ID.

dbOpenQueryFile

Opens an output file for the output from *dbQuery*.

Syntax

`dbOpenQueryFile "fileName"`

where the arguments are as follows:

fileName

Name of file to open

Valid values: Name of any query file in the current directory

License Required

Milkyway

dbQuery

Prints the content of the record.

Syntax

`dbQuery libraryOrCellId "recordName"`

where the arguments are as follows:

libraryOrCellId

Database object ID of the cell or library to which information applies

Valid values:

- Cell ID of any cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

- Library ID of the library currently open in write (W) mode
- (dbGetCurrentLibId)

For more information, see “Specifying Libraries” on page 3-10.

recordName

Name of the record to print

Valid values: Name of any record in the specified library or cell

License Required

Milkyway

Extensions for Modifying the Database

This section provides syntax information for extensions that provide the ability to modify the Synopsys databases.

Note:

Unless otherwise specified, the examples are from Astro.

dbAddCellInstToCellRegion

Adds a cell instance to a cell region. This is equivalent to assigning a specific cell instance to a region. The instance's visible location does not change a result of this function.

Syntax

`dbAddCellInstToCellRegion cellId regionId cellInstId`

where the arguments are as follows:

cellId

Database object ID of cell to which region information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

regionId

Database object ID of the region to which cell instance information applies

Valid values: Region ID of any region in the specified cell

cellInstId

Database object ID of the cell instance to add

Valid values: Cell instance ID of any cell instance in the specified cell

License Required

Milkyway

Related Commands

dbGetCellInstByName (see page 3-107)

dbDeleteCellInstFromCellRegion (see page 3-189)

dbAddObjectToGroup

Adds an object to a group.

Syntax

dbAddObjectToGroup *cellId groupID objectId position*

where the arguments are as follows:

cellId

Database object ID of the cell to which group information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

groupId

Database object ID of the group to which object information applies

Valid values: Group ID of any group in the cell

objectId

Database object ID of the object to add to the group

Valid values: Object ID of any object within the cell

Note:

Objects do not have to be homogeneous within the group. For example, you could place a net and a cell together in one group.

position

A positive integer specifying the position of the object in the group. This is only meaningful in an ordered group.

License Required

Milkyway

Related Commands

`aprCreateGroup` (see page 52 of the *IC Implementation Command Reference Manual*) to add cells to a cell group

`dbDeleteObject` (see page 3-192)

`dbPurgeObject` (see page 3-236)

`dbRemoveObjectFromGroup` (see page 3-242)

dbClearCellEEQClass

Clears the electrical-equivalent class in the specified library.

Note:

Astro does not currently use the electrical-equivalent class.

Syntax

```
dbClearCellEEQClass libId
```

where *libId* is the database object ID of any library in the current directory from which you want to delete electrical-equivalent class.

License Required

Milkyway

Related Commands

[dbCreateCellEEQClass \(see page 3-142 \)](#)

[dbClearCellEQClass \(see page 3-132 \)](#)

[dbVerifyCellEEQClass \(see page 3-297 \)](#)

dbClearCellEQClass

Clears both electrical-equivalent and logical-equivalent classes in the specified library.

Note:

Astro does not currently use the electrical-equivalent class.

Syntax

```
dbClearCellEQClass libId
```

where *libId* is the database object ID of any library in the current directory from which you want to delete both the electrical-equivalent and logical-equivalent classes.

License Required

Milkyway

Related Commands

[dbCreateCellEEQClass](#) (see page 3-142)

[dbCreateCellLEQClass](#) (see page 3-144)

[dbClearCellEEQClass](#) (see page 3-131)

[dbClearCellLEQClass](#) (see page 3-133)

[dbVerifyCellEEQClass](#) (see page 3-297)

[dbVerifyCellLEQClass](#) (see page 3-298)

dbClearCellLEQClass

Clears logical-equivalent class in the specified library.

Syntax

```
dbClearCellLEQClass libId
```

where *libId* is the database object ID of any library currently open from which you want to delete logical-equivalent class.

License Required

Milkyway

Related Commands

`dbCreateCellLEQClass` (see page 3-144)

`dbClearCellLEQClass` (see page 3-133)

`dbVerifyCellLEQClass` (see page 3-298)

dbConnect

Connects any of the following to a net:

- Port
- Port instance
- Wire
- Contact
- Path

Use `dbConnectPortInstByName` (see page 3-136) to create connections of port instances to nets that are not members of the net. Generally, power and ground connections are not members of the net.

Side Effects

Because the netlist is changed and no ECO is recorded, care should be taken when you use this function.

Syntax

`dbConnect cellId netId memberId`

where the arguments are as follows:

cellId

Database object ID of the cell to which net information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

netId

Database object ID of the net to which information applies

Valid values: Net ID of any net in the specified cell

memberId

Database object ID of the net member to connect

Valid values: Object ID of any object connected to the specified net

`topCellName`

Name of the top level design cell

License Required

Milkyway

dbConnectPortInstByName

Connects a port instance of a cell instance to a net. The port instance must currently be unconnected to a net.

Syntax

```
dbConnectPortInstByName cellId netId instName portInstName  
topCellName
```

where the arguments are:

cellId Database object ID of the cell to which port information applies.

Valid values:

Cell ID of any open cell in the current library (geGetEditCell). For information, see “Specifying Cells” on page 3-57.

netId Database object ID of the net to which port information applies.

Valid values:

Net ID of any net in the cell.

instName Name of the cell instance to which port information applies.

Valid values:

Name of any cell instance in the cell.

portInstName Name of the port instance to connect to a net.

Valid values:

Name of any port in the cell.

topCellName Name of the top level design cell.

Related Commands

[dbDisconnectPortInstByName \(see page 3-199 \)](#)

License Required

Milkyway

dbConvertRegionToMacroCell

Copies the content of a cell region to a macro cell. This function does not alter the existing cell region.

Syntax

```
dbConvertRegionToMacroCell "cellName" regionNumber  
"macroCellName"
```

where the arguments are as follows:

cellName

Name of cell from which cell region content is to be converted

Valid values: Name of any cell in the current library

regionNumber

Number of the cell region to convert

Valid values: Any positive integer corresponding to a region in the cell

macroCellName

Name of macro cell to which cell region content is to be copied

Valid values: Any string with 31 characters or less

License Required

Milkyway

dbCreateCellBoundary

Creates a cell boundary in a cell. If a cell boundary already exists for the cell, the existing boundary will be replaced by the new one.

Syntax

```
dbCreateCellBoundary cellId pointArray
```

where the arguments are as follows:

cellId

Database object ID of the cell to which cell boundary information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

pointArray

Coordinates indicating the dimensions of the cell boundary

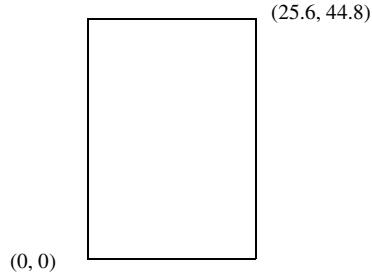
Valid syntax:

```
' ((x1 y1) (x2 y2) (x3 y3) (x4 y4) (x1 y1))
```

Example

```
dbCreateCellBoundary cellId
'((0 0) (0 44.8) (25.6 44.8) (25.6 0) (0 0))
#t
```

This example creates the following cell boundary.



License Required

Milkyway

dbCreateCellBoundaryForCell

Creates a cell boundary in a cell. If a cell boundary already exists for the cell, the existing boundary will be replaced by the new one.

Syntax

dbCreateCellBoundary ForCell cellName pointArray

where the arguments are as follows:

cellName

Name of the master cell of the soft macro where you want to create a boundary.

pointArray

Coordinates indicating the dimensions of the cell boundary

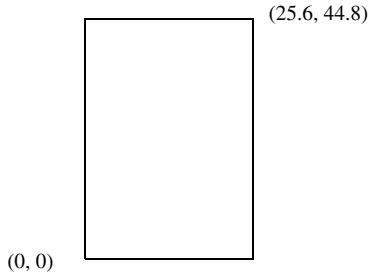
Valid syntax:

' ((x_1 y_1) (x_2 y_2) (x_3 y_3) (x_4 y_4) (x_1 y_1))

Example

```
dbCreateCellBoundary ForCell cellName  
'((0 0) (0 44.8) (25.6 44.8) (25.6 0) (0 0))  
#t
```

This example creates the following cell boundary.



License Required

Milkyway

dbCreateCellIEQClass

Defines electrically-equivalent pairs of GDSII Stream cells. If you set a pair of cells electrically-equivalent, both cells are electrically-equivalent to all other cells set electrically-equivalent to either of the two.

Note:

Astro does not currently use the electrical-equivalent class.

Syntax

```
dbCreateCellIEQClass libId "cellName1" "cellName2"
```

where the arguments are as follows:

libId

Database object ID of the library in which you want to create the electrical-equivalent class

Valid values: Library ID of any library in the current directory

cellName1

Name of the cell you want to set electrically-equivalent to another cell, *cellName2*, (and to all other cells set electrically-equivalent to *cellName2*)

Valid values: Name of any cell in the specified library

cellName2

Name of the second cell you want to set electrically-equivalent to another cell, *cellName1*, (and to all other cells set electrically-equivalent to *cellName1*)

Valid values: Name of any cell, other than *cellName1*, in the specified library

License Required

Milkyway

Related Commands

dbClearCellEEQClass (see page 3-131)

dbClearCellEQClass (see page 3-132)

dbVerifyCellEEQClass (see page 3-297)

dbCreateCellLEQClass

Defines logically equivalent pairs of GDSII Stream cells. If you set a pair of cells logically equivalent, both cells are logically equivalent to all other cells set logically equivalent to either of the two.

Syntax

```
dbCreateCellLEQClass libId "cellName1" "cellName2"
```

where the arguments are as follows:

libId

Database object ID of the library in which you want to create a logical-equivalent class

Valid values: Library ID of any library in the current directory

cellName1

Name of the cell you want to set logically equivalent to another cell, *cellName2*, (and to all other cells set logically equivalent to *cellName2*)

Valid values: Any cell in the specified library

cellName2

Name of the cell you want to set logically equivalent to another cell, *cellName1*, (and to all other cells set logically equivalent to *cellName1*)

Valid values: Any cell, other than *cellName1*, in the specified library

Related Commands

[dbClearCellLEQClass \(see page 3-133 \)](#)

`dbClearCellEQClass` (see page 3-132)

`dbVerifyCellLEQClass` (see page 3-298)

Also, see “Matrices for Timing Functions” on page A-1 of the *Static Timing Analyzer User Guide* for a list of timing commands.

License Required

Milkyway

dbCreateCellProperty

Creates a cell property.

Note:

A Synopsys application infers the property type from the property value. List values are illegal. Only atoms may be used for the argument value.

Syntax

`dbCreateCellProperty cellId "propName" propValue`

where the arguments are as follows:

cellId

Database object ID of the cell to which you want to add the cell property

Valid values:

- Cell ID of any open cell in the current library
- (`geGetEditCell`)

For information, see “Specifying Cells” on page 3-57.

propName

Name you want to assign to the new cell property

Valid values: Any text string with 31 characters or less

propValue

Value you want to assign to the new cell property

Valid value types:

- Boolean
- Float
- Integer
- String

License Required

Milkyway

Related Commands

`dbDeletePropertyByName` (see page 3-197)

dbCreateCellRegion

Creates a cell region with horizontal rows.

Syntax

`dbCreateCellRegion cellId pointArray`

where the arguments are as follows:

cellId

Database object ID of the cell to which cell you want to add the cell region

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

pointArray

Coordinates indicating the dimensions of the cell region

Valid syntax:

' ((x1 y1) (x2 y2))

License Required

Milkyway

Related Commands

dbSetCellRegionBBox (see page 3-261)

dbSetCellRegionTargetPercent (see page 3-262)

dbAddCellInstToCellRegion (see page 3-128)

dbCreateCellRow (see page 3-150)

dbDeleteObject (see page 3-192)

dbPurgeObject (see page 3-236)

`dbReplaceRegionWithInst` (see page 3-245)

dbCreateCellRow

Creates a cell row.

Syntax

`dbCreateCellRow cellId regionId pointArray`

where the arguments are as follows:

cellId

Database object ID of the cell containing the cell region to which you want to add the cell row

Valid values:

- Cell ID of any open cell in the current library
- (`geGetEditCell`)

For information, see “Specifying Cells” on page 3-57.

regionId

Database object ID of the cell region to which you want to add the cell row.

Valid values: Region ID of any region in the specified cell

pointArray

Coordinates indicating the dimensions of the cell row

Valid syntax:

`' ((x1 y1) (x2 y2))`

License Required

Milkyway

Related Commands

dbSetCellRowMirror (see page 3-263)

dbStartCellRowDoubleBack (see page 3-286)

dbCreateChildNet

Creates a child net of the specified net. Child nets are used when a net has connections in more than one region, or in a region and macro cells or pad cells.

Caution!

Synopsys product specialists recommend that you do not use this function. It is essential that the naming conventions be consistent with those used by the Synopsys application for proper system operation. Failure to follow the naming conventions could cause the database problems.

Syntax

`dbCreateChildNet cellId "childNetName" netId`

where the arguments are as follows:

cellId

Database object ID of the cell to which you want to add the child net

Valid values:

- Cell ID of any open cell in the current library
- (`geGetEditCell`)

For information, see “Specifying Cells” on page 3-57.

childNetName

Name you want to assign to the new child net

Valid values: The parent net name with the string “*nn*” when *nn* is the region number the net will be used in.

netId

Database object ID of the existing (parent) net from which you want to create the child net

Valid values: Net ID of any net in the master cell

License Required

Milkyway

Related Commands

`dbGetNetByName` (see page 3-117)

`dbPurgeNet` (see page 3-234)

`dbPurgeNetByName` (see page 3-235)

dbCreateChipPort

Creates a top-level port on a cell. The function `dbCreateChipPort` returns *nil* (a set of parentheses ()) when it succeeds. When `dbCreateChipPort` fails, the Synopsys application returns an error message. For example, if a port already exists for the specified net, Astro displays the following error message:

```
Port creation failed because WARNING: Port already  
exists for the given net.  
( )
```

If you plan to do timing-driven placement and your design does not have top-level ports, you need to create them before you partition the cells.

Note:

After creating the port, set its characteristics using dbSetCellPortTypes (see page 3-257).

Syntax

```
dbCreateChipPort "libName" "cellName" "netName" "portName"
```

where all the arguments are as follows:

libName

Name of the library containing the cell to which you want to add the top-level port

Valid values: Name of any library in the current directory

cellName

Name of the cell to which you want to add the port

Valid values: Name of any cell in the specified library

netName

Name of the net to which you want the new port connected

Valid values: Name of any net in the specified cell

portName

Name you want to assign to the new port

Valid values: Any string with 1023 characters or less

Example

```
dbCreateChipPort "demo" "topj" "VDD" "VDD"  
( )
```

This example creates a top-level port on cell *topj* in library *demo*. The port is named *VDD* and is connected to the *VDD* net.

License Required

Milkyway

Related Commands

`dbSetPortDirection` (see page 3-281)

dbCreateContact

Creates a contact.

Prerequisites

All contacts used in your design must be defined in the *deviceTable* section of the technology file. For more information, see “Technology File” on page 1 of the *Milkyway Data Preparation User Guide*.

Syntax

```
dbCreateContact cellId contactNumber point
```

where the arguments are as follows:

cellId

Database object ID of the cell for which you want to create a contact

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

contactNumber

Number you specified in the *deviceTable* section of the technology file that identifies the contact you want to create.

Valid values: Any integer from 1 to 255 used in the *deviceTable* to identify a contact. For the following *deviceTable* section, the *contactNumber* for the contact, *polymet1*, is 1.

*deviceTable



```
*contact 1 polymet1
cont poly m1 1 1 1.2 2.5 2.0
defaultRowNumber 2
parasitics 0.2 0.0 0.0
*end
```

point

Location of the new contact

Valid syntax:

$'(x\ y)$

License Required

Milkyway

dbCreateError

Signals an LVS or DRC error for a specified class of errors. If successful, *dbCreateError* returns the database object ID of your error.

Caution!

Synopsys LVS and DRC do not check for user-defined errors.

Prerequisite

You must first define a class of errors using *dbCreateErrorClass* (see page 3-159).

Syntax

```
dbCreateError cellId errClassId "message"
```

where the arguments are as follows:

cellId

Database object ID of the cell containing a specified class of errors for which you want to signal an error

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

errClassId

Database object ID of the class of errors for which you want to signal an error

Valid values: Error class ID of any class of errors in the specified cell

message

Message describing the signaled error

Valid values: Any string with 200 characters or less

Note:

When executing dbCreateError, assign the result to a variable.

License Required

Milkyway

Related Commands

dbSetErrorStatus (see page 3-268)

dbCreateErrorClass (see page 3-159)

dbCreateErrorClass

Creates a class of LVS or DRC errors. If successful, *dbCreateErrorClass* returns the database object ID of your class of errors.

Caution!

Synopsys LVS and DRC do not check for user-defined errors.

Syntax

```
dbCreateErrorClass cellId errLevel "className" "message"
```

where the arguments are as follows:

cellId

Database object ID of the cell for which you want to create a class of errors

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

errLevel

Number you want to assign to the severity of the error

Valid values: Positive integer from 0 to 4

className

Name you want to assign to the new class of errors

Valid values: Any string with 31 characters or less

message

Message describing the class of error

Valid values: Any string with 200 characters or less

Note:

When executing dbCreateErrorClass, assign the result to a variable.

License Required

Milkyway

Related Commands

`dbSetErrorClassPattern` (see page 3-266)

`dbSetErrorClassStatus` (see page 3-267)

`dbCreateError` (see page 3-157)

dbCreateGroup

Creates a group of objects. If successful, `dbCreateGroup` returns the database object ID of your group.

Syntax

`dbCreateGroup cellId "groupName" groupNumber`

where the arguments are as follows:

cellId

Database object ID of the cell for which you want to create a group

Valid values:

- Cell ID of any cell in the open library
- (`geGetEditCell`)

For information, see “Specifying Cells” on page 3-57.

groupName

Name you want to assign to the new cell group

Valid values: Any string with 31 characters or less

Note:

Group names are always case sensitive, regardless of the case sensitivity setting.

groupNumber

Number you want to assign to the new group

Valid values: A number between 0 and 255. A Synopsys application reserves group numbers 0 - 200, and 255 for predefined purposes.

Note:

When executing dbCreateGroup, assign the result to a variable.

License Required

Milkyway

Related Commands

dbDeleteObject (see page 3-192)

dbPurgeObject (see page 3-236)

dbAddObjectToGroup (see page 3-129)

dbRemoveObjectFromGroup (see page 3-242)

dbGetAllInstFromGroup (see page 3-106)

aprCreateGroup (see page 52) to create cell groups

dbCreateHierNet

Creates a hierarchical net.

Syntax

```
dbCreateHierNet cellId "netName" netId cellRegionNumber
```

where the arguments are as follows:

cellId

Database object ID of the cell for which you want to add a hierarchical net

Valid values:

- Cell ID of any cell in the open library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

netName

Name you want to assign to the new hierarchical net

Valid values: The parent net name with the string “*nn*” appended, where *nn* is the cell region number.

netId

Database object ID of the parent net

Valid values: Net ID of the any net in the cell

cellRegionNumber

Number of the region in which net is placed

Valid values: A number between 0 and 255

License Required

Milkyway

dbCreateHorizontalWire

Creates an internal horizontal wire.

Syntax

```
dbCreateHorizontalWire cellId masterWireId wireLength
xPoint
yPoint
```

where the arguments are as follows:

cellId

Database object ID of the cell to which wire information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

masterWireID

Database object ID of the master wire record

Valid values: Wire ID of any wire in the specified cell

wireLength

Length of the wire

Valid types of values:

- A positive integer extends the wire to the right of the origin.
- A negative integer extends the wire to the left of the origin.

xPoint

X coordinate of the location where you want to place the wire

Valid values: Any integer

yPoint

Y coordinate of the location where you want to place the wire

Valid values: Any integer

License Required

Milkyway

dbCreateLeqPortGroup

Creates a logically equivalent set of ports for a cell. Only one set of logically equivalent ports may be defined at this time.

Syntax

```
dbCreateLeqPortGroup cellId `("port1" "port2" ...)
```

where the arguments are as follows:

cellId

Database object ID of the cell to which the node information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

portlist

The list of ports to be defined as logically equivalent

License Required

Milkyway

Related Commands

dbDeleteLeqPortGroup (see page 3-191)

dbCreateVerticalWire

Creates an internal vertical wire.

Syntax

```
dbCreateVerticalWire cellId masterWireId wireLength xPoint  
                  yPoint
```

where the arguments are as follows:

cellId

Database object ID of the cell to which wire information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

masterWireId

Database object ID of the master wire record

Valid values: Wire ID of any wire in the specified cell

wireLength

Length of the wire

Valid values:

- A positive integer extends the wire above the origin.
- A negative integer extends the wire below the origin.

xPoint

X coordinate of the location where you want to place the wire

Valid values: Any integer

yPoint

Y coordinate of the location where you want to place the wire

Valid values: Any integer

License Required

Milkyway

dbCreateNet

Creates a net in a cell. When successful, *dbCreateNet* returns the database object ID of the new net.

Syntax

`dbCreateNet cellId "netName"`

where the arguments are as follows:

cellId

Database object ID of the cell to which net information applies

Valid values:

- Cell ID of any open cell in the current library

- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

netName

Name of the net to create

Valid values: Any text string 1023 characters or less

Example -

Create a new net *VDD* and merge all members from net *Logic1* into *VDD*.

```
define cell1 (dbOpenCell "control.Net1" "w")
db:2
define vddNetId (dbCreateNet cell1 "VDD")
db:110034
dbMoveAllNetMember cell1 (dbGetNetByName cell1 "Logic1")
vddNetId
dbCloseCell cell1
#t
```

In this example,

- The first line opens the cell *control.Net1* and assigns the database ID to *cell1*.
- The second line creates a new net, *VDD*, and assigns the netId to *vddNetId*.
- The third line merges all members from *Logic1* into the new net, *VDD* and deletes the empty net *Logic1*.
- The fourth line closes the cell *control.Net1*.

License Required

Milkyway

Related Commands

`dbGetNetByName` (see page 3-117) for another example of
`dbCreateNet`

`dbMoveAllNetMember` (see page 3-225) for another example of
`dbCreateNet`

dbCreateNode

Creates a graph node in a cell.

Syntax

```
dbCreateNode cellId "type" nodeNo center edgeNoUp edgeNoDown  
          edgeNoLeft edgeNoRight
```

where the arguments are as follows:

cellId

Database object ID of the cell to which the node information applies

Valid values:

- Cell ID of any open cell in the current library
- (`geGetEditCell`)

For information, see “Specifying Cells” on page 3-57.

type

Type of cell design

Valid values:

- *singleRegion* specifies a single-region design.

- *multipleRegion* specifies a multiple-region design.

Note:

This field is case-insensitive.

nodeNo

Number of the graph node

Valid value: A positive integer

center

Location of the center of the node

Valid syntax:

' (x y)

edgeNoUp

Number of the edge above the node

Valid value: A positive integer

edgeNoDown

Number of the edge below the node

Valid value: A positive integer

edgeNoLeft

Number of the edge to the left of the node

Valid value: A positive integer

edgeNoRight

Number of the edge to the right of the node

Valid value: A positive integer

License Required

Milkyway

dbCreateParamSet

Creates a parameter set record in a cell.

Syntax

```
dbCreateParamSet cellId "appName"
```

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

appName

Name of the set to create

Valid values: Any text string 31 characters or less

License Required

Milkyway

dbCreateParentNet

Creates a parent net in a cell.

Caution!

Synopsys product specialists recommend that you do not use this function.

Syntax

```
dbCreateParentNet cellId "parentNetName" netId  
cellRegionNumber
```

where the arguments are as follows:

cellId

Database object ID of the cell to which net information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

parentNetName

Name of the net to create

Valid values: Any text string 1023 characters or less

netId

Database object ID of an existing net from which a parent net is created

Valid values: Net ID of any net in the specified cell

cellRegionNumber

Number of the region to which parent net is added

Valid values: A number between 0 to 255, but typically 0

License Required

Milkyway

dbCreatePin

Creates a pin record for a cell.

Syntax

```
dbCreatePin cellId netId "pinName" "layerName" pointArray
```

where the arguments are as follows:

cellId

Database object ID of the cell to which pin information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

netId

Database object ID of the net to which pin information applies

Valid values: Net ID of any net in the specified cell

pinName

Name of pin to create

Valid values: Any text string 31 characters or less

layerName

Name of layer to which pin information applies

Valid values: Name of any layer defined in the cell library's technology file.

pointArray

Coordinates indicating the size and shape of the pin

Valid syntax:

```
' ((x1 y1) (x2 y2))
```

The first point is the lower left coordinate of the box and the second point is the upper right coordinate of the box. A polygon pin may be specified by entering a set of 3 or more points where the last point is the same as the first.

License Required

Milkyway

dbCreatePolygon

Creates a polygon record for a cell.

Syntax

```
dbCreatePolygon cellId "layerName" pointArray
```

where the arguments are as follows:

cellId

Database object ID of the cell to which polygon information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

layerName

Name of layer to which polygon information applies

Valid values: Name of any layer in the specified cell

pointArray

Coordinates indicating the size and shape of the polygon

Valid syntax:

((x₁ y₁) (x₂ y₂) ... (x₁ y₁))

A closed set of points.

License Required

Milkyway

dbCreatePort

Creates a port record on a specified net.

Syntax

dbCreatePort *cellId* *netId* "portName"

where the arguments are as follows:

cellId

Database object ID of the cell to which port information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

netId

Database object ID of the net to which port information applies

Valid values: Net ID of any net in the specified cell

portName

Name of port to create and attach to the specified net

Valid values: Any text string with 1023 characters or less

License Required

Milkyway

dbCreatePreRoute

Creates a preroute record in a cell.

Syntax

```
dbCreatePreRoute cellId netId startPt endPt edgeNo width  
length layerNumber offset
```

where the arguments are as follows:

cellId

Database object ID of the cell to which preroute information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

netId

Database object ID of the net to which preroute information applies

Valid values: Net ID of any net in the specified cell

startPt

Point corresponding to the beginning of the preroute segment

Valid syntax:

' (x y)

endPt

Point corresponding to the end of the preroute segment

Valid syntax:

' (x y)

edgeNoEdge to which preroute information applies

Valid values: Any positive integer

width

Width of the preroute segment in user units

Valid values: Any positive integer

length

Length of the preroute segment between pins

Valid values: Any positive integer specifying the distance between the preroute segment and pin

layerNumber

Number of cell layer where preroute record should be placed

Valid values: Any positive integer

offset

Distance between the preroute segment and the center of the node

Valid values: Any positive integer

License Required

Milkyway

dbCreateProperty

Creates a property record associated with an object. The type of the property is inferred from the value.

Syntax

```
dbCreateProperty cellId objectId "propName" value
```

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

objectId

Database object ID of the object to which property information applies

Valid values: Object ID of any object in the specified cell

propName

Name of property record to create and associate object with

Valid values: Any text string 35 characters or less

value

Value assigned to the property

Valid types:

- String
- Float
- Integer
- Boolean

License Required

Milkyway

dbCreatePseudoPath

Creates a pseudo path record in a cell.

Syntax

```
dbCreatePseudoPath cellId width "layerName" pointArray
```

where the arguments are as follows:

cellId

Database object ID of the cell to which pseudo path information applies

Valid values:

- Cell ID of any open cell in the current library
- (`geGetEditCell`)

For information, see “Specifying Cells” on page 3-57.

width

Width of the pseudo path in user units

layerName

Name of layer where pseudo path record is to be placed

pointArray

Coordinates indicating the shape and size of the pseudo path

Valid syntax:

`' ((x1 y1), (x2 y2), ... , (xn yn))`

License Required

Milkyway

dbCreateRectangle

Creates a rectangle record in a cell.

Syntax

`dbCreateRectangle cellId "layerName" pointArray`

or

`dbCreateRectangle cellId layerNumber pointArray`

where the arguments are as follows:

cellId

Database object ID of the cell to which rectangle information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

layerName

Name of layer to which rectangle information applies

Valid values: Name of any layer in technology file

layerNumber

Number of cell layer to which rectangle information applies

Valid values: Positive integer

pointArray

Coordinates indicating the shape and size of the rectangle

Valid syntax:

‘((X1 Y1) (X2 Y2))

The first point is the lower left coordinate of the rectangle and the second point is the upper right coordinate of the rectangle.

License Required

Milkyway

dbCreateRegionPin

Creates a region pin record in a cell. If no region port exists, a side effect of this function is to create the region port. Region pins cannot exist without an associated region port.

Syntax

```
dbCreateRegionPin cellId netId regionNetId regionId  
"regionPinName" "layerName" pointArray
```

where the arguments are as follows:

cellId

Database object ID of the cell to which region pin information applies.

Valid values:

- Cell ID of any open cell in the current library.
- (geGetEditCell).

For information, see “Specifying Cells” on page 3-57.

netId

Database object ID of the net to which region pin information applies.

Valid values: Net ID of any net in the specified cell

regionNetId

Database object ID of the region net to which region pin information applies.

Valid values: Net ID of any region net in the current library.

regionId

Database object ID of the region to which region pin information applies.

Valid values: Region ID of any region in the current library.

regionPinName

Name of region pin to create.

Valid values: Any text string with 31 characters or less.

layerName

Name of layer to which region pin information applies.

Valid values: Name of any layer in the current library's technology file.

pointArray

Coordinates indicating the shape and size of the region pin.

Valid syntax:

' ((*x1 y1*) , (*x2 y2*) , ... (*xn yn*))

License Required

Milkyway

dbCreateRegionPort

Creates a region port record on a cell.

Syntax

dbCreateRegionPort *cellId netId regionNetId regionId*

where the arguments are as follows:

cellId

Database object ID of the cell to which region port information applies.

Valid values:

- Cell ID of any open cell in the current library.
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

netId

Database object ID of the net to which region port information applies.

Valid values: Net ID of any net in the specified cell.

regionNetId

Database object ID of the region net to which region port information applies.

Valid values: Net ID of any region net in the current library.

regionId

Database object ID of the region to which region port information applies.

Valid values: Region ID of any region in the current library.

License Required

Milkyway

dbCreateText

Creates a text record in a cell.

Syntax

```
dbCreateText cellId "layerName" "just" "rotation" "mirror"  
"height origin "text"
```

where the arguments are as follows:

cellId

Database object ID of the cell to which the text information applies.

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

layerName

Name of layer to which the text information applies

Valid values: Name of any layer in the specified cell library’s technology file.

just

Placement of the text around the origin.

Valid values:

ll places the text to the lower left of the origin.

lc places the text to the lower center of the origin.

lr places the text to the lower right of the origin.

cl places the text to the center left of the origin.

cc centers the text around the origin.

cr places the text to the center right of the origin.

ul places the text to the upper left of the origin.

uc places the text to the upper center of the origin.

ur places the text to the upper right of the origin.

rotation

Degree to which the text should be rotated.

Valid values:

0

90

180

270

mirror

Indicates the orientation of the text.

Valid values:

No

X

Y

height

Size of text in the y direction specified in user units.

origin

Coordinate to which text should be anchored.

Valid syntax:

$'(x1\ y1)$

text

Text string that should appear in the cell.

Valid values: Any string.

License Required

Milkyway

dbCreateWireMaster

Creates a wire master record in a specified cell.

Syntax

```
dbCreateWireMaster cellId pathType layerType wireWidth  
layerNumber
```

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

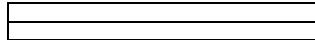
For information, see “Specifying Cells” on page 3-57.

pathType

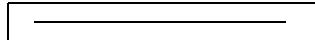
Alignment of the wire

Valid values:

- 0 means that the path edge aligns with the coordinate.



- 2 means that the path edge extends beyond the coordinate.



layerType

0

Note:

This field is currently unused

wireWidth

Width of the wire in database units

layerNumber

Layer number for instances of this wire master

License Required

Milkyway

dbDeleteCellInstFromCellRegion

Deletes a cell instance's association with a cell region. The cell instance will not change its location. Only the relationship indicating this cell instance is to be placed inside the cell region is removed.

Syntax

```
dbDeleteCellInstFromCellRegion cellId regionId cellInstId
```

dbDeleteCellInstFromCellRegion

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

regionId

Database object ID of the region to which cell instance information applies

Valid values: Region ID of any region in the specified cell

cellInstId

Database object ID of the cell instance to delete

Valid values: Cell instance ID of any cell instance in the specified region

License Required

Milkyway

dbDeleteCellRowDoubleBack

Resets cell row placement orientation.

Note:

Use prior to executing dbStartCellRowDoubleBack (see page 3-286).

Syntax

`dbDeleteCellRowDoubleBack cellId regionNum`

where the arguments are as follows:

cellId

Database object ID of the cell to which cell row information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

regionNum

Number of the cell region in which cell row information applies

Valid values: Positive integers that identify regions inside the specified cell

License Required

Milkyway

dbDeleteLeqPortGroup

Deletes the current logical equivalent port information.

Syntax

`dbDeleteLcqPortGroup cellId`

where the arguments are as follows:

cellId

Database object ID of the cell to which cell row information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

License Required

Milkyway

Related Commands

`dbDeleteLcqPortGroup` (see page 3-191)

dbDeleteObject

Deletes an object from a cell.

Syntax

`dbDeleteObject cellId objectId`

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies.

Valid values:

- Cell ID of any open cell in the current library.
- (geGetEditCell).

For information, see “Specifying Cells” on page 3-57.

objectId

Database object ID of the object to delete.

Valid values: Object ID of any object in the specified cell. For example:

- cell instance ID
- net ID
- pin ID
- port ID
- property ID
- region pin ID
- region port ID

Note:

This is only a partial list.

License Required

Milkyway

dbDeleteOneLayerRectangle

Deletes all rectangles on a specified layer.

Syntax

```
dbDeleteOneLayerRectangle "libName" layerNumber
```

where the arguments are as follows:

libName

Name of the library to which layer information applies

Valid values: Name of any library in the current directory

layerNumber

Number of the layer from which all rectangles are to be deleted

Valid values: Any positive integer corresponding to a layer in the specified library

License Required

Milkyway

dbDeletePin

Deletes a pin record in a cell. Function performs the same functionality as dbPurgeObject (see page 3-236).

Syntax

```
dbDeletePin cellId pinId
```

where the arguments are as follows:

cellId

Database object ID of the cell to which pin information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

pinId

Database object ID of the pin to delete

Valid values: Pin ID of any pin in the specified cell

License Required

Milkyway

dbDeleteProperty

See dbDeletePropertyById (see page 3-196)

dbDeletePropertyById

Deletes a specified property record.

Syntax

`dbDeletePropertyById cellId propertyId`

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies

Valid values:

- Cell ID of any open cell in the current library

- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

propertyId

Database object ID of the property to delete

Valid values: Object ID of any property in the specified cell

License Required

Milkyway

dbDeletePropertyByName

Deletes a specified property record from a cell.

Syntax

```
dbDeletePropertyByName cellId objectID "propName"
```

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies.

Valid values:

- Cell ID of any open cell in the current library.
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

objectID

Database object ID of the object to which information applies.

Valid values: Object ID of any object in the specified cell. For example:

- cell instance ID
- net ID
- pin ID
- port ID
- region pin ID
- region port ID

Note:

This is only a partial list.

propName

Name of property record to delete.

Valid values: Name of any property in the specified cell.

License Required

Milkyway

dbDisconnect

Disconnects an object from a net. Use dbDisconnectPortInstByName (see page 3-199) to erase connections of port instances to nets that are not members of the net. Generally, power and ground connections are not members of the net.

Side Effects

The netlist is changed and no ECO is recorded. Use care when using this function.

Syntax

```
dbDisconnect cellId netId memberId
```

where the arguments are as follows:

cellId

Database object ID of the cell to which net information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

netId

Database object ID of the net to which information applies

Valid values: Net ID of any net in the specified cell

memberId

Database object ID of the net member to disconnect

Valid values: Object ID of any object connected to the specified net

topCellName

Name of the top level design cell

License Required

Milkyway

dbDisconnectPortInstByName

Disconnects power or signal connections from port instances and associated pin instances. This function can be used to disconnect power and ground connections that are not members of the net.

Side Effects

This function changes the netlist. No ECO is recorded.

Syntax

```
dbDisconnectPortInstByName cellId "netName" "cellInstName"  
"portInstName" "topCellName"
```

where the arguments are as follows:

cellId

Database object ID of cell to which port instance information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

netName

Name of the net to which port instance information applies

Valid values: Name of any net in the specified cell

cellInstName

Name of the cell instance to which port instance information applies

Valid values: Name of any cell instance in the specified cell

portInstName

Name of the port instance to which information applies

Valid values: Name of any port instance associated with the specified cell instance

topCellName

Name of the top level design cell

License Required

Milkyway

dbDisplayCellRailProperty

Displays the y coordinates (in database units) of the edges of the power/ground rails in a standard cell from the bottom up. The standard cell should be opened in .FRAM view.

Syntax

`dbDisplayCellRailProperty cellId`

where the arguments are as follows:

cellId

Database object ID of the cell to which the y coordinate applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

Related Command

`dbUpdateCellRailProperty` (see page 3-290) updates the y coordinate of an edge of a power/ground rail.

License Required

Milkyway

dbDumpCellPlacement

Dumps the cell placement information to a file.

Syntax

```
dbDumpCellPlacement cellId "cellType" "fileName"
```

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies

Valid values:

- Cell ID of any open cell in the current library
- (`geGetEditCell`)

For information, see “Specifying Cells” on page 3-57.

cellType

Type of cells that placement information is dumped

Valid values:

- *macro* means that placement information for macro cells dumps.

- *standard* means that placement information for standard cells dumps.
- *pad* means that placement information for pad cells dumps.
- *cornerPad* that placement information for corner pad cells dumps.

fileName

Name of file to which cell placement information dumps

Valid values: Any text string

License Required

Milkyway

dbDumpGPortTable

Writes the library global port information to a file. To specify port information, use dbSetCellPortTypes (see page 3-257).

Syntax

```
dbDumpGPortTable "libName" "fileName"
```

where the arguments are as follows:

libName

Name of the library from which you want to write the global port information

Valid values: Name of any library in the current directory

fileName

Name of file to which you want to write the information

Valid values: Any text string

License Required

Milkyway

dbDumpSingleViewPageNumber

Dumps the database page number of specified cell in the current opened library with the specified view name.

Syntax

```
dbDumpSingleViewPageNumber cellId "viewName"
```

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

viewName

Name of view from which cell content dumps

Valid values: Name of any view associated with the specified cell

- *.cel*
- *.fram*
- *.netl*

- .exp

License Required

Milkyway

dbDumpTechFile

Creates a technology file that you can edit.

Syntax

```
dbDumpTechFile libId "outFile" dumpAllFlag
```

where the arguments are as follows:

libId

Database object ID of the library to which technology file information applies

Valid values: Library ID of any library in the current directory

outFile

Name of technology file, to create

Valid values: Any text string with a file extension of .tf.

dumpAllFlag

Whether or not all technology information dumps

Valid values:

- #t means that all technology information, including predefined color, stipple, and so forth, dumps.
- #f means that only user defined information dumps.

Example

```
dbDumpTechFile lib1 "train2out" #t  
ž#t
```

This example dumps the technology information of a library *lib1* to an editable ASCII file.

License Required

Milkyway

Related Commands

dbReplaceTechFile (see page 3-46)

dbDumpViewRecordNumber

Dumps the database record number of every cell in the current opened library with the specified view name.

Syntax

```
dbDumpViewRecordNumber "viewName"
```

where *viewName* is the name of the view from which you want the database record of each cell. The *viewName* can be the name of any view in the current library. Valid values are:

- *.cel*
- *.fram*
- *.netl*
- *.exp*

License Required

Milkyway

dbDumpViewPageNumber

Dumps the database page number of every cell in the current opened library with the specified view name.

Syntax

```
dbDumpViewPageNumber "viewName"
```

where *viewName* is the name of the view from which you want the database page number of each cell. The *viewName* can be the name of any view in the current library. Valid values are:

- *.cel*
- *.fram*
- *.netl*
- *.exp*

License Required

Milkyway

dbForceRecompCellIBBox

Forces the system to update all database page bBox of the specified cell, and then update the cell bBox.

Syntax

```
dbForceRecompCellBBox cellId
```

where the *cellId* is the database object ID of the cell for which you want to update the bBox. The argument *cellId* can be either of the following;

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

License Required

Milkyway

dbForceRecursiveUpdateAllHierarchy

This command operates within cell hierarchy and propagates any changes to top level cells. It updates ALL cells. It is similar to dbForceUpdateHierarchy on page 3-211 , except dbForceUpdateHierarchy only works on the top cell’s immediate children.

Syntax

```
dbForceRecursiveUpdateAllHierarchy cellId
```

where the *cellId* is the database object ID of the cell for which you want to update. The argument *cellId* can be either of the following;

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

Example

Cell A has an instance B, and the master cell of B has an instance C. If the master cell in C has changed, you must do *dbForceRecursiveUpdateAllHierarchy* to propagate the change to the top level for all cells.

License Required

Milkyway

dbForceUpdateHierarchy

Forces the system to update all cell instances of the specified cell to make them consistent.

Syntax

`dbForceUpdateHierarchy cellId`

where the *cellId* is the database object ID of the cell. The argument *cellId* can be either of the following:

- Cell ID of any open cell in the current library
- (*geGetEditCell*)

For information, see “Specifying Cells” on page 3-57.

License Required

Milkyway

db-foreach

Executes a statement for each object associated with a sub-object in the way specified by the scan name.

Syntax

```
db-foreach libraryOrCellId subObjectIdOrNil fieldName  
"scanName"  
loopVariable (statements)
```

where the arguments are as follows:

libraryOrCellId

Database object ID of the cell or library to which information applies

Valid values:

- Cell ID of any cell in the current library
- Library ID of any library in the current directory

subObjectIdOrNil

Database object ID of the object from which information is desired or *nil*

Valid values:

- Object ID of any object within the specified library or cell
- *nil* ('()) if the scan is directly associated with the library or cell

FieldName

Name of a valid scan. If the first character of *scanName* is '?', the command will display all valid scan names associated with *subObjectId* and/or *libraryOrCellId*.

scanName

Name of a valid scan. Valid values for *scanName* depend on the values of the *libraryOrCellId* and *subObjectIdOrNil* fields. Use dbFetchScanTypes (see page 3-101) for a complete list.

These values apply for a given library ID when *subObjectIdOrNil* is equal to *nil* ('()).

- *CellName*
- *CellVersion*
- *RefLibrary*

These values apply for a given cell ID when *subObjectIdOrNil* is equal to *nil* ('()).

- *Trapezoid*
- *Polygon*
- *Rectangle*
- *Path*
- *Text*
- *HorizontalWire*
- *VerticalWire*
- *WireMaster*

- *Contact*
- *ContactArray*
- *PseudoPath*
- *PseudoContactArray*
- *Net*
- *NetHeader*
- *Pin*
- *Port*
- *CellInstHeader*
- *CellInstMaster*
- *CellInst*
- *PortInstMaster*
- *PortInst*
- *CellInstArray*
- *ProtectionFrame*
- *CellRegion*
- *CellRow*
- *HardFence*
- *SoftFence*
- *RegionPin*
- *RegionPort*

- *GateSite*
- *GateSiteArray*
- *WireTrack*
- *WireDir*
- *ErrorClass*
- *Error*
- *ErrorObject*
- *AttachedFile*
- *Node*
- *Edge*
- *Channel*
- *ChannelGate*
- *ChannelGatePin*
- *PreRoute*
- *InterRouteSeg*
- *PortToPort*
- *PortToPortTLU*
- *State*
- *StateTransition*
- *StateCommandClass*
- *StateCommand*

- *PointArray*
- *CellLocArray*
- *PadLocArray*
- *CellEQClass*
- *SlicingTreeNode*
- *Guide*
- *ChannelGuide*
- *Property*
- *PropertyHeader*
- *PropertyAttribute*
- *Group*
- *GroupMember*
- *PlanGroup*
- *ParameterSet*
- *DelayTable*
- *DelayValueTable*
- *UserUnformatted*

The following values are applicable to AstroGA only.

- *InternalSOGXWire*
- *InternalSOGYWire*
- *SOGWireMaster*

- *SOGContact*
- *SOGContactMaster*

These values apply for a given cell ID when *subObjectldOrNil* is not equal to *nil* ('()).

<i>subObjectld</i> Type	<i>scanName</i>
Cell boundary	Valid value: <i>Property</i>
Cell EQ class	Valid value: <i>Property</i>
Cell instance	Valid values: <ul style="list-style-type: none"> • <i>PortInst</i> • <i>Property</i>
Cell instance array	Valid value: <i>Property</i>
Cell instance header	Valid value: <i>CellInst</i>
Cell instance master	Valid values: <ul style="list-style-type: none"> • <i>CellInst</i> • <i>PortInstMaster</i>
Cell name	Valid value: <i>Property</i>
Cell region	Valid values: <ul style="list-style-type: none"> • <i>CellRow</i> • <i>Property</i> • <i>RegionPin</i> • <i>RegionPort</i>
Cell row	Valid values: <ul style="list-style-type: none"> • <i>CellInst</i> • <i>Property</i>
Cell version	Valid value: <i>Property</i>
Channel	Valid value: <i>Property</i>

subObjectid Type	scanName
Channel gate	Valid values: <ul style="list-style-type: none">• <i>ChannelGatePin</i>• <i>Property</i>
Channel gate pin	Valid value: <i>Property</i>
Channel guide	Valid value: <i>Property</i>
Delay table	Valid value: <i>Property</i>
Edge	Valid value: <i>Property</i>
Gate site	Valid value: <i>Property</i>
Gate site array	Valid value: <i>Property</i>
Group	Valid values: <ul style="list-style-type: none">• <i>GroupMember</i>• <i>Property</i>
Guide	Valid value: <i>Property</i>
Hard fence	Valid values: <ul style="list-style-type: none">• <i>CellInst</i>• <i>Group</i>• <i>Property</i>
Net	Valid values: <ul style="list-style-type: none">• <i>ChildNet</i>• <i>NetMember</i>• <i>Property</i>
Node	Valid value: <i>Property</i>
Parameter set	Valid value: <i>Property</i>
Path	Valid value: <i>Property</i>
Pin	Valid value: <i>Property</i>
Polygon	Valid value: <i>Property</i>

subObjectid Type	scanName
Port	Valid values: <ul style="list-style-type: none">• <i>Pin</i>• <i>Property</i>
Port to port delay	Valid value: <i>Property</i>
Port to port table	Valid value: <i>Property</i>
Preroute segment	Valid value: <i>Property</i>
Protection frame	Valid value: <i>Property</i>
Pseudo path	Valid value: <i>Property</i>
Reference library	Valid value: <i>Property</i>
Region pin	Valid value: <i>Property</i>
Region port	Valid values: <ul style="list-style-type: none">• <i>Property</i>• <i>Region Pin</i>
Soft fence	Valid values: <ul style="list-style-type: none">• <i>CellInst</i>• <i>Group</i>• <i>Property</i>
Slicing tree node	Valid value: <i>Property</i>
Text	Valid value: <i>Property</i>
Wire track	Valid value: <i>Property</i>
Wire direction	Valid value: <i>Property</i>

loopVariable

Name of variable to be bound to a new location and assigned a database ID for each pass through the loop. The field *loopVariable* persists only for the life of the loop.

statement

Any single statement. The statement may be a *begin* statement if multiple function calls are needed. If *statement* returns *nil*, the loop is aborted. This is equivalent to a break in C.

Caution!

You should avoid creating objects that would be a part of the scan during the scan. The new object may appear in the current scan.

Example 1 - List the names of all the cells contained in a library.

```
define libName "ex4"
"ex4"
define libId (dbOpenLib libName)
db:1
(db-foreach libId '() "cellName" cellNameRec
(begin
(display (dbFetchObjectField libId cellNameRec
"cellName" ))
#t)
)
`()
dbCloseLib
#t
```

In the example,

- The fist line creates an alias, *libName*, for the library *ex4*.
- The second line creates an alias, *libId*, for the database object ID of *ex4*.
- The third line displays the value of the cell name field for each cell name record in the library *ex4*.

Example 2

```
define libId (dbOpenLib "DESIGN")
db-foreach libId '() "?" () ()
    CellName
    CellVersion
    RefLibrary
```

Example 3

```
define cellId (geGetEditCell)
define netId (dbGetNetByName cellId "VDD")
b-foreach cellId netId "?" () ()
    ChildNet
    PortInst
    Port
    NetMember
    NetWiring
```

License Required

Milkyway

Related Commands

dbFetchScanTypes (see page 3-101) lists the scan types available for the specified library or cell ID/subobject ID pair.

dbMakeWireMasterId

Compose a real object ID from a cell ID and a *internWireMasterId*. The database uses *internWireMasterId* when storing wires to save space in the database. When fetching a wire's *masterId*, the real object ID of the master must be composed with this function.

Syntax

```
dbMakeWireMasterId cellId layer internWireMasterId
```

where the arguments are as follows:

cellId

ID of the cell from which wire is selected

Valid values:

Cell ID of any open cell in the current library. If the cell is open, you can substitute the following for *cellId*

(geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

layer

Number of the layer for which you want to fetch the wire master ID. If you do not specify a layer, this command returns the wire master ID in the first 256 created wire masters regardless of the layer.

Valid values: routing layer

internWireMasterID

The wire’s *internWireMasterId*

Example - Compose a wire master ID. (Assume a wire is selected)

```
define wireMasterId (dbMakeWireMasterId (geGetEditCell)
(dbFetchObjectField (geGetEditCell)
(car (geGetSelectedSet 1))
"internWireMasterId")
)
db:18b0002
```

In this example, you compose a wire master ID and assign the ID to the variable *wireMasterId*.

To get information about the wire master, use dbFetchObject (see page 3-95).

```
dbFetchObject (geGetEditCell) wireMasterId  
  ("filler" 0) ("layerNumber" 5) ("wireWidth" 200)  
  ("status" 0) ("pathType" 0) ("layerType" 0)  
  ("recordType" "WireMaster"))
```

The function *dbFetchObject* lists all the information contained in the wire master record.

License Required

Milkyway

dbMarkCellType

Sets the layout cell subtype for the specified cell.

Syntax

```
dbMarkCellType libName cellName layoutSubType
```

where the arguments are as follows:

libName

Library that contains the cell

Valid values:

- Any existing library name and path

cellName

Cell name you want to set.

Valid values:

- Any existing cell name in the library

layoutSubType

Layout cell subtype

Valid values:

- a string in the set {"padCell", "cornerCell", "moduleCell", "standardCell", "macroCell", "imageCell", "coverCell", "otherCell"}

Example

```
dbMarkCellType "libName" "cellName" "macroCell"
dbMarkCellType
"/home/user/libname" "cellName" "imageCell"
```

License Required

Milkyway

dbMilkywayTranslateString

This command translates a Milkyway name into a verilog name.

Due to verilogs' restrictions on escaping special characters like '[]' or the hierarchy separator '/', you may want to query what the object name is in verilog. *dbMilkywayTranslateString* is used for this query.

A library name is required since different libararies may have different bus naming style, such as [] vs. <>. They will affect the translation result.

Note:

dbMilkywayTranslateString is query only, no changes are made to the database.

Syntax

```
dbMilkywayTranslateString libName objNameInMilkyway
```

where the arguments are as follows:

libName

Library that contains the cell

Valid values:

- Any existing library name and path

License Required

Milkyway

dbMoveAllNetMember

Moves all members from one net to another. Region ports and region pins are not modified.

Note:

Because this function ignores the parent/child net relationship, do not use this function after you generate region pins for multiple-region designs in Astro.

Syntax

```
dbMoveAllNetMember cellId fromNetId toNetId
```

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

fromNetId

Database object ID of the net from which information is to be copied

Valid values:

- Net ID of any net in the specified cell
- (dbGetNetByName cellId “netName”)

For information, see dbGetNetByName on page 3-117.

toNetId

Database object ID of the net to which information is to be copied

Valid values:

- Net ID of any net in the specified cell
- (dbGetNetByName cellId “netName”)

For information, see dbGetNetByName (see page 3-117).

Example 1 -

Merge all members of net *Logic0* into net *VSS*.

```
define cell1 (dbOpenCell "control.Net1" "w")
db:2
dbMoveAllNetMember cell1 (dbGetNetByName cell1 "Logic0")
(dbGetNetByName cell1 "VSS")

dbCloseCell cell1
#t
```

In this example,

- The first line opens the cell *control.Net1* and assigns the object ID to *cell1*.
- The second lines merges all members of net *Logic0* into net *VSS*.
- The third line saves the changes you made and closes *cell1*.

Example 2 -

Merge tiehi/tielo nets into **powerNetName** and **groundNetName**.

```
define libraryName "designs"
"designs"
define expandedCellName "top.Net1"
"top.Net1"
define powerNetName "VDD"
"VDD"
define groundNetName "VSS"
"VSS"
dbOpenLib libraryName
db:1
define cell1 (dbOpenCell expandedCellName "w")
db:2
dbMoveAllNetMember cell1 (dbGetNetByName cell1 "Logic1")
(dbGetNetByName cell1 powerNetName)
dbMoveAllNetMember cell1 (dbGetNetByName cell1 "Logic0")
(dbGetNetByName cell1 groundNetName)
dbCloseCell cell1
#t
dbCloseLib
#t
```

In this example,

- The first four lines create global variables.
- The fifth line opens the library *designs*.
- The sixth line opens the expanded cell *top.Net1*.

- The seventh line merges the tiehi nets from net *Logic1* to *powerNetName*.
- Line eight merges the tielo nets from net *Logic0* to *groundNetName*.
- The ninth line saves the changes you made and closes *cell1*.
- The tenth line saves and closes the library *designs*.

Note:

If the nets VDD and VSS did not exist prior to this example, you could delete lines three and four, and substitute the following lines for lines seven and eight:

```
dbMoveAllNetMember cell11 (dbGetNetByName cell11 "Logic1")
(dbCreateNet cell11 powerNetName)
dbMoveAllNetMember cell11 (dbGetNetByName cell11 "Logic0")
(dbCreateNet cell11 groundNetName)
```

Related Commands

`dbCreateNet` (see page 3-167) for another example of *dbMoveAllNetMember*.

License Required

Milkyway

dbMoveCellOrigin

Changes the origin of a cell to a point you specify by coordinates. If successful, *dbMoveCellOrigin* returns *#t*.

Syntax

```
dbMoveCellOrigin cellId origin
```

where the arguments are as follows:

cellId

Database object ID of the cell to which origin information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

origin

Origin of cell

Valid syntax:

' (x y)

License Required

Milkyway

dbPurgeAllEmptyNet

Purges all empty nets in the specified cell.

Syntax

dbPurgeAllEmptyNet *cellId*

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

License Required

Milkyway

dbPurgeAllSingleNet

Purges all single port (or port instance) nets in the specified cell.

Syntax

`dbPurgeAllSingleNet cellId`

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

License Required

Milkyway

dbPurgeCellInstByName

Purges a cell instance record from database.

Syntax

```
dbPurgeCellInstByName cellId "cellInstName"
```

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

cellInstName

Name of cell instance record to purge

Valid values: Name of any cell instance in the specified cell

License Required

Milkyway

dbPurgeCellInstMaster

Purges a cell instance master record from database. A side effect of this function is to purge all cell instances of this master.

Syntax

```
dbPurgeCellInstMaster cellId "cellInstMasterName"
```

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

cellInstMasterName

Name of cell instance master record to purge

Valid values: Name of any cell instance in the specified cell

License Required

Milkyway

dbPurgeNet

Purges a specified net record from database.

Syntax

```
dbPurgeNet cellId netId
```

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

netId

Database object ID of the net to purge

Valid values: Net ID of any net in the specified cell

License Required

Milkyway

dbPurgeNetByName

Purges a specified net record from database.

Syntax

```
dbPurgeNetByName cellId "netName"
```

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

netName

Name of the net to purge

Valid values: Name of any net in the specified cell

License Required

Milkyway

dbPurgeObject

Purges an object from database. (same as dbDeleteObject (see page 3-192)).

Syntax

`dbPurgeObject cellId objectId`

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies

Valid values: Cell ID of any cell in the current library

objectId

Database object ID of the object to be deleted

Valid values: Object ID of any object in the specified cell. For example,

- *netId*
- *portId*
- *portInstId*

License Required

Milkyway

dbPurgePin

Purges a pin record from database.

Syntax

`dbPurgePin cellId pinId`

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

pinId

Database object ID of the pin to purge

Valid values: Pin ID of any pin in the specified cell

License Required

Milkyway

dbPurgePort

Purges a port record from database.

Syntax

```
dbPurgePort cellId portId
```

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

portId

Database object ID of the port to purge

Valid values: Port ID of any port in the specified cell

License Required

Milkyway

dbPurgeRegionPin

Purges a region pin record from database.

Syntax

```
dbPurgeRegionPin cellId regionPinId
```

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

regionPinId

Database object ID of the region pin to purge

Valid values: Region pin ID of any port in the specified cell

License Required

Milkyway

dbPurgeRegionPort

Purges a region port record from database.

Syntax

`dbPurgeRegionPort cellId regionPortId`

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies.

Valid values:

- Cell ID of any open cell in the current library.
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

regionPortId

Database object ID of the region port to purge.

Valid values: Region port ID of any port in the specified cell.

License Required

Milkyway

dbRebuildLib

dbRebuildLib will rebuild a library (lib file) by scanning all the cells in the library directory. This command will not create a library property or attached files.

Syntax

`dbRebuildLib "libName"`

where the arguments are as follows:

libName

The name of the library

License Required

Milkyway

dbRemoveObjectFromGroup

Removes an object from a group.

Syntax

`dbRemoveObjectFromGroup cellId groupId objectId`

where the arguments are as follows:

cellId

Database object ID of the cell to which group information applies

Valid values:

- Cell ID of any open cell in the current library
- (`geGetEditCell`)

For information, see “Specifying Cells” on page 3-57.

groupId

Database object ID of the group from which object is to removed

Valid values: Group ID of any group in the current cell

objectId

Database object ID of the object to be removed

Valid values: Object ID of any member of the specified group. For example,

- *netId*
- *pinId*
- *portId*
- *portInstId*

License Required

Milkyway

dbReplaceCellInst

Replaces the cell instance in a cell. Connections to the instance will be maintained if the port names of the new cell master match the port names of the existing cell master. Ports in the old master that have no equivalent port in the new master are disconnected. Ports in the new master that have no equivalent in the old master are left unconnected.

Syntax

```
dbReplaceCellInst cellId instId "newCellName" "newViewName"  
"topCellName"
```

where the arguments are as follows:

cellId Database object ID of the master cell to which cell instance information applies.

Valid values:

ID of any cell in the open library. If the cell is open, you can substitute the following for *cellId*: (*geGetEditCell*). For more information, see “Specifying Cells” on page 3-57.

instId Database object ID of the cell instance to which information applies.

Valid values:

Cell instance ID of any cell instance in the current library.

newCellName Name of master cell to replace current master cell.

Valid values:

Name of any master cell in the current library.

newViewName View name of master cell to replace current master cell.
me Valid values:
Name of any master view in the current library.

topCellName Name of the top cell in the open cell.

Example

```
dbReplaceCellInst (geGetEditCell) 25619 "INVX3" ".FRAM"  
"top"
```

License Required

Milkyway

dbReplaceRegionWithInst

Replaces a cell region with a cell instance.

Syntax

```
dbReplaceRegionWithInst "cellName" regionNumber  
"macroCellName" "cellInstName"
```

where the arguments are as follows:

cellName

Name of the cell to which cell instance information applies

Valid values: Name of any cell in the current library

regionNumber

Number of the region that is to be replaced

Valid values: Any positive integer

macroCellName

Name of macro cell that is to be the cell instance's child cell

Valid values: Name of any macro cell in the current library

cellInstName

Name of cell instance that replaces the cell region

Valid values: Name of any cell instance in the current cell

License Required

Milkyway

dbResetCellMustJoinPin

Removes external connection specifications for pins on a port.

Application

This command applies to the applications in the following table.

Application	Tool: <i>Menu Command</i>
Astro/AstroGA	none

Syntax

`dbResetCellMustJoinPin cellId portname`

where the arguments are

cellId

ID of the pin/blockage (*.fram*) cell for which you want to remove specifications for external pin connections. If the cell is open, you can substitute the following for *cellId*:

(geGetEditCell)

portName

Name of the port for which you want to remove specifications for external pin connections

Example

```
dbResetCellMustJoinPin (geGetEditCell) "A"
```

This function removes the specifications for external pin connections on port A in the open cell.

Related Commands

dbSetCellMustJoinPin (see page 1712 of the *IC Implementation Command Reference Manual*) instructs the application to externally connect together specified pins on a port.

dbSetCellMustJoinAllPin (see page 1710 of the *IC Implementation Command Reference Manual*) instructs the application to externally connect together all pins on a specified port (or all ports with names matching a specified pattern).

dbCreateMustJoinNets (see page 1588 of the *IC Implementation Command Reference Manual*) creates a net for each port in the specified cell that has must-join pins not assigned to any net.

License Required

Milkyway

dbSetCellInstECOStatus

Sets the ECO status bits in the cell instance *statusFlag* field.

Syntax

```
dbSetCellInstECOStatus cellId cellInstId "ECOStatus"
```

where the arguments are as follows:

cellId

Database object ID of the master cell to which cell instance information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

cellInstId

Database object ID of the cell instance to which ECO information applies

Valid values: Instance ID of any cell instance in the current cell

ECOStatus

ECO status of the cell instance

Valid values:

- *ECOReset* resets all the ECO status bits.

- *ECODelete* resets the *dbbPRCellInstECODelete* bit. This indicates that the ECO placer should delete the cell instance during ECO placement.
- *ECOAdd* sets the *dbbPRCellInstECOAdd* bit. This indicates that the cell was newly added by an ECO, and that the ECO placer must place it. After the ECO placer places the cell, this bit is reset.
- *ECORReplace* sets the *dbbPRCellInstECORReplace* bit. This indicates that the cell was replaced by an ECO.
- *ECOIgnore* sets the *dbbPRCellInstECOIgnore* bit. This bit is automatically set by dbCreateCellLEQClass (see page 3-144), geAddCell (see page 2071 of the *IC Implementation Command Reference Manual*), and TDF *insertPad* to indicate that this cell was interactively added by the user and should not be deleted even if it does not appear in the ECO netlist.

License Required

Milkyway

dbSetCellInstECOSatusByName

Sets the ECO status bits in the cell instance *statusFlag* field.

Syntax

```
dbSetCellInstECOSatus cellId cellInstName "ECOSatus"
```

where the arguments are as follows:

cellId

Database object ID of the master cell to which cell instance information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

cellInstName

Name of the cell instance to which ECO information applies

Valid values: Name of any cell instance in the current directory

ECOStatus

ECO status of the cell instance

Valid values:

- *ECOReset* resets all the ECO status bits.
- *ECODelete* resets the *dbbPRCellInstECODelete* bit. This indicates that the ECO placer should delete the cell instance during ECO placement.
- *ECOAdd* sets the *dbbPRCellInstECOAdd* bit. This indicates that the cell was newly added by an ECO, and that the ECO placer must place it. After the ECO placer places the cell, this bit is reset.
- *ECOReplace* sets the *dbbPRCellInstECOReplace* bit. This indicates that the cell was replaced by an ECO.

- *ECOIgnore* sets the *dbbPRCellInstECOIgnore* bit. This bit is automatically set by *dbCreateCellLEQClass* (see page 3-144), *geAddCell* (see page 2071 of the *IC Implementation Command Reference Manual*), and TDF *insertPad* to indicate that this cell was interactively added by the user and should not be deleted even if it does not appear in the ECO netlist.

License Required

Milkyway

dbSetCellInstFixStatusByName

Specifies the placement status of a cell instance.

Syntax

```
dbSetCellInstFixStatusByName cellId cellInstName
    "placementStatus"
```

where the arguments are as follows:

cellId

Database object ID of the master cel*I to which cell instance information applies

Valid values:

- Cell ID of any open cell in the current library
- (*geGetEditCell*)

For information, see “Specifying Cells” on page 3-57.

cellInstName

Name of the cell instance to which placement information applies

Valid values: Instance name of any cell instance in the current directory with the exception of I/O pad instances

ECOStatus

Placement status of the cell instance. For information, see `aprCmdFixCell` on page 178 of the *IC Implementation Command Reference Manual*

Valid values:

- Unplaced
- Placed
- Fixed
- RigidFixed

License Required

Milkyway

dbSetCellInstOrigin

Establishes the origin of a cell instance.

Syntax

```
dbSetCellInstOrigin cellId "cellInstName" origin
```

where the arguments are as follows:

cellId

Database object ID of the master cell to which cell instance information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

cellInstName

Name of the cell instance to which origin information applies

Valid values: Name of any cell instance in the current library

origin

Origin of the cell instance

Valid values:

- (list (x y))

License Required

Milkyway

dbSetCellInstStatus

Specifies the status of a cell instance.

Syntax

```
dbSetCellInstStatus cellId cellInstId "status"
```

where the arguments are as follows:

cellId

Database object ID of the master cell to which cell instance information applies

Valid values:

- Cell ID of any open cell in the current library
- (`geGetEditCell`)

For information, see “Specifying Cells” on page 3-57.

cellInstId

Database object ID of the cell instance to which status information applies

Valid values: *instanceId* of any instance in the current cell

status

Placement status of the cell instance

Valid values:

- *Unplaced*
- *Placed*
- *Fixed*
- *RigidFixed*

License Required

Milkyway

dbSetCellOrigin

Changes the origin of a cell to a point you specify by predefined points. If successful, `dbSetCellOrigin` returns `#t`.

Syntax

`dbSetCellOrigin cellId "justification"`

where the arguments are as follows:

cellId

Database object ID of the cell to which the origin information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

justification

Location of the new cell origin

Valid values:

- *ll* the lower left position of the cell’s cell boundary.
- *lc* the lower center position of the cell’s cell boundary.
- *lr* the lower right position of the cell’s cell boundary.
- *cl* the center left position of the cell’s cell boundary.
- *cc* the center position of the cell’s cell boundary.
- *cr* the center right position of the cell’s cell boundary.
- *ul* the upper left position of the cell’s cell boundary.
- *uc* the upper center position of the cell’s cell boundary.
- *ur* the upper right position of the cell’s cell boundary.

License Required

Milkyway

dbSetCellPortTypeExtraView

Specifies the extra view name, with the exception of CEL view, FRAM view, or TIM view, before using dbSetCellPortTypes.

Syntax

```
dbSetCellPortTypeExtraView "viewname" . . .
```

The number of the view name must be less than three (3).

where the arguments are:

<i>viewName</i>	Name of the view which you want to use in the command dbSetCellPortTypes. Valid values: Can either be “SMASH” or other view name.
-----------------	---

Example

```
dbSetCellPortTypeExtraView "SMASH" "BARR"
```

Related Commands

dbSetCellPortTypes (see page 3-257)

License Required

Milkyway

dbSetCellPortTypes

Specifies the following information for ports in a cell:

- Whether a port is input, output, inout, or tristate.
- Whether a port is connected to a power, ground, or clock net.
- Whether any unconnected ports in a cell should tie to power or ground.
- Whether a port asynchronous with respect to clock is active high or active low.
- Whether a port is the input or output to a clocked cell.
- Whether a port is the address input of a memory cell (RAM or ROM).
- Whether a port is the enable for the scan clock.
- Whether a port is a scan input, scan output, or scan clock input for a scannable cell.
- Whether a port is a control line for a tristate cell.
- Whether a port is for connecting to a flip chip bump.

Some of this information is stored in the database in the .FRAM cell, and some of this information is stored as a library attached file. This function does *not* store any information in the .CEL cell. Use dbSetPortDirection (see page 3-281) instead.

For more information, see “Specifying Port Information for Advanced Operations” on page 4-14 of the *Milkyway Data Preparation User Guide*.

Syntax

```
dbSetCellPortTypes "libName" "cellName" ' ( { ( "portName"  
{ "type"} ...} ...) ) append?
```

where the arguments are as follows:

libName Name of the library containing the cell to which port information applies.

Valid values:

Name of any library in the current directory.

cellName Name of the cell to which port information applies.

Valid values:

Name of any cell in the specified library.

portName Name of the port to which information applies.

Valid values:

Name of any port in the specified cell.

type	<p>Type that applies to the specified port.</p> <p>Valid values:</p> <p>For any port, you can specify one or more of the following:</p> <ul style="list-style-type: none"> • Input means that the port is an input port. • Output means that the port is an output port. • Inout means that the port is an inout port. • Tristate means that the port is a tristate port. • Power means that the port should be connected to a power net. • Ground means that the port should be connected to a ground net. • Clock means that the port should be connected to a clock net. • Tieup means that if the port is not connected, it should be tied to power. • Tiedown means that if the port is not connected, it should be tied to ground. • Async_rising means that the port is asynchronous with respect to clock and is active high. • Async_falling means that the port is asynchronous with respect to clock and is active low. • DataIn means that the port is the input to a clocked cell. • DataOut means that the port is the output from a clocked cell.
type (cont.)	<ul style="list-style-type: none"> • AddressIn means that the port is the address input of a memory cell (RAM or ROM). • Enable means that the port is the enable for the scan clock. • FlipChip means the port is for connecting to a bump. • ScanIn means that the port is a scan input for a scannable cell. • ScanOut means that the port is a scan output for a scannable cell. • ScanClock means that the port is a scan clock input for a scannable cell. • TristateDisable means that the port is a control line for a tristate cell.

Note:

If you change the tie-up/tie-down information after floorplanning, you must update connections in the cells using aprPGConnect (see page 100 of the *IC Implementation Command Reference Manual*).

append?

Whether or not the information should be appended to information specified by previous dbSetCellPortTypes functions.

Valid values:

- #t means that Astro should append any existing information with the new information.
- #f means that Astro should replace any existing information with the new information.

Note:

To write current port information to a file, use dbDumpGPortTable on page 3-203.

Example

```
dbSetCellPortTypes "demo" "AND4" '(  
  ("VDD" "Inout" "Power")  
  ("C" "Input" "Tieup")  
  ("A" "Input" "Tieup")  
  ("B" "Input" "Tieup")  
  ("D" "Input" "Tieup")  
  ("X" "Output")  
  ("GND" "Inout" "Ground")  
) #f
```

This example provides port information for ports *VDD*, *C*, *A*, *B*, *D*, *X*, and *GND* in cell *AND4*, which is in the library *demo*. The *#f* indicates that the information should replace any existing port information for the cell *AND4*.

Related Commands

[dbSetCellPortTypeExtraView](#)

License Required

Milkyway

dbSetCellRegionBBox

Specifies a bBox for a cell region.

Syntax

```
dbSetCellRegionBBox cellId regionId pointArray
```

where the arguments are as follows:

cellId

Database object ID of the cell to which the cell region information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

regionId

Database object ID of the region to which the information applies

Valid values: Region ID of any region in the specified cell

pointArray

Coordinates of the *bBox* specified in the following format:

```
' ((x1 y1) (x2 y2))
```

The first point is the lower left corner and the second point is the upper right corner.

Example

```
define_cell (getEditCell)
define_cellinst (dbGetCellInstByName_cell "block1/inst2")
```

```
define_cellRegion  
(dbGetRegionIdByCellInstId_cell_cellinst)  
dbSetCellRegionBBox_cell_region '((100 100) (200 200))
```

License Required

Milkyway

dbSetCellRegionTargetPercent

Specifies the target percent of a region.

Syntax

```
dbSetCellRegionTargetPercent cellId regionId targetPercent
```

where the arguments are as follows:

cellId

Database object ID of the cell to which the cell region information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

regionId

Database object ID of the region to which the information applies

Valid values: Region ID of any region in the specified cell

targetPercent

Percent of cells to be placed in this region

Valid values: Any positive integer less than 15000. (Note the region percent is scaled by 100)

License Required

Milkyway

dbSetCellRowMirror

Specifies the placement orientation of a cell row.

Syntax

`dbSetCellRowMirror cellId cellRowId orient`

where the arguments are as follows:

cellId

Database object ID of the cell to which the cell row information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

cellRowId

Database object ID of the cell row to which the information applies

Valid values: Cell row ID of any cell row in the specified cell

orient

Whether or not the cell row should be mirrored

Valid values:

- `#t` mirrors the cell row.
- `#f` leaves the cell row unmodified.

License Required

Milkyway

dbSetCellType

Specifies the type of a cell.

Syntax

```
dbSetCellType cellId "cellType"
```

where the arguments are as follows:

cellId

Database object ID of the cell to which the information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

cellType

Type that applies to the cell

Valid values:

- *maskLayout*
- *netList*

- *schematic*
- *symbolic*
- *pcbLayout*
- *behavior*
- *logicModel*
- *graphic*
- *document*
- *stranger*
- *unknown*

License Required

Milkyway

dbSetColorClassColor

Specifies the color of an error class.

Syntax

`dbSetColorClassColor cellId errClassId color`

where the arguments are as follows:

cellId

Database object ID of the cell to which the error information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

errClassId

Database object ID of the error class to which information applies

Valid values: Error class ID of any error class in the specified cell

color

Color that applies to the error

Valid values: An integer from 0 to 255 that indexes the color map

License Required

Milkyway

dbSetErrorClassPattern

Specifies the pattern of an error class.

Syntax

`dbSetErrorClassPattern cellId errClassId pattern`

where the arguments are as follows:

cellId

Database object ID of the cell to which error information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

errClassId

Database object ID that must be an error class ID

Valid values: Error class ID of any error class in the specified cell

pattern

Pattern that applies to the error

Valid values: An integer from 0 to 255 that indexes the fill pattern map

License Required

Milkyway

dbSetErrorClassStatus

Specifies the status of an error class.

Syntax

```
dbSetErrorClassStatus cellId errClassId status
```

where the arguments are as follows:

cellId

Database object ID of the cell to which error information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

errClassId

Database object ID of the error class to which information applies

Valid values: Error class ID of any error class in the specified cell

status

Level that applies to the error

Valid values:

- 0 gives an informational error.
- 1 gives a warning.
- 2 gives an error.
- 3 gives a severe error.

License Required

Milkyway

dbSetErrorStatus

Specifies the status of an error.

Syntax

```
dbSetErrorStatus cellId errId status
```

where the arguments are as follows:

cellId

Database object ID of the cell to which error information applies

Valid values:

- Cell ID of any open cell in the current library

- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

errId

Database object ID of error to which information applies

Valid values: Error class ID of any error class in the specified cell

status

Level that applies to the error

Valid values:

- 0 gives an informational error.
- 1 gives a warning.
- 2 gives an error.
- 3 gives a severe error.

License Required

Milkyway

dbSetLayoutSubType

Specifies the layout subtype of a cell.

Syntax

```
dbSetLayoutSubType cellId "layoutSubType"
```

where the arguments are as follows:

cellId

Database object ID of cell to which information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

layoutSubType

Layout subtype that applies the specified cell

Valid values:

- ioPad
- padFillerCell
- cornerPad
- standardCell
- standardCell2
- stdFillerCell
- macro
- chip
- image
- cover
- unknown

License Required

Milkyway

dbSetMaxMetalRoutingLayer

This command allows you to control the routing layer usage by setting the maximum metal routing layer.

The default value of the “maxMetalRoutingLayer” will be set during floorplanning to the highest metal in the techfile. You can check this using “dbGetMaxMetalRoutingLayer” on page 3-36.

Note:

Use only three or four layers of metal in a five or six layer library.

Syntax

```
dbSetMaxMetalRoutingLayer cellId metal_num
```

where the arguments are as follows:

cellId

Database object ID of cell to which net information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

metal_num

Number of layers of metal in a metal routing layer.

Example

To set the maximum metal routing layer:

```
(dbSetMaxMetalRoutingLayer (geGetEditCell) 4)  
-> will route up to 'metal4'
```

License Required

Milkyway

dbSetNetType

Specifies the type of a net.

Syntax

```
dbSetNetType cellId netId "netType"
```

where the arguments are as follows:

cellId

Database object ID of cell to which net information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

netId

Database object ID of net to which information applies

Valid values: Net ID of any net in the specified cell

netType

Type that applies to the specified net

Valid values:

- *power* means that the net is a power net.

- *ground* means that the net is a ground net.
- *clock* means that the net is a clock net.
- *signal* means that the net is a signal net.

License Required

Milkyway

dbSetNetTypeByName

Specifies the type of a net.

Syntax

```
dbSetNetTypeByName cellId "netName" "netType"
```

where the arguments are as follows:

cellId

Database object ID of cell to which net information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

netName

Name of the net to which information applies

Valid values: Name of any port in the specified cell

netType

Type that applies to the specified net

Valid values:

- *power* means that the net is a power net.
- *ground* means that the net is a ground net.
- *clock* means that the net is a clock net.
- *signal* means that the net is a signal net.

License Required

Milkyway

dbSetPinAccessDir

Specifies the access direction of a pin.

Syntax

```
dbSetPinAccessDir cellId pinId "accessDir"
```

where the arguments are as follows:

cellId

Database object ID of the cell to which pin information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

pinId

Database object ID of the pin to which information applies

Valid values: Pin ID of any pin in the specified cell

accessDir

Direction you want Astro to use when routing the pin

Valid values:

- *L* means that the pin is only accessible from the left.
- *R* means that the pin is only accessible from the right.
- *U* means that the pin is only accessible from above.
- *D* means that the pin is only accessible from below.
- *LR* means that the pin is accessible from the left and right.
- *LU* means that the pin is accessible from the left and above.
- *LD* means that the pin is accessible from the left and below.
- *RU* means that the pin is accessible from the right and above.
- *RD* means that the pin is accessible from the right and below.
- *UD* means that the pin is accessible from above and below.
- *LRUD* means that the pin is accessible from all directions.
- *LRU* means that the pin is not accessible from below.
- *LRD* means that the pin is not accessible from above.
- *LUD* means that the pin is not accessible from the right.
- *RUD* means that the pin is not accessible from the left.

License Required

Milkyway

dbSetPinAccessDirByName

Specifies the access direction of a pin.

Syntax

```
dbSetPinAccessDirByName cellId "pinName" "accessDir"
```

where the arguments are as follows:

cellId

Database object ID of the cell to which pin information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

pinName

Name of pin to which information applies.

Valid values: Name of any pin in the specified cell

accessDir

Direction you want Astro to use when routing the pin

Valid values:

- *L* means that the pin is only accessible from the left.
- *R* means that the pin is only accessible from the right.
- *U* means that the pin is only accessible from above.
- *D* means that the pin is only accessible from below.

- *LR* means that the pin is accessible from the left and right.
- *LU* means that the pin is accessible from the left and above.
- *LD* means that the pin is accessible from the left and below.
- *RU* means that the pin is accessible from the right and above.
- *RD* means that the pin is accessible from the right and below.
- *UD* means that the pin is accessible from above and below.
- *LRUD* means that the pin is accessible from all directions.
- *LRU* means that the pin is not accessible from below.
- *LRD* means that the pin is not accessible from above.
- *LUD* means that the pin is not accessible from the right.
- *RUD* means that the pin is not accessible from the left.

License Required

Milkyway

dbSetPinLayerNumber

Specifies the layer number of a pin and creates a pin.

Syntax

```
dbSetPinLayerNumber cellId pinId layerNumber
```

where the arguments are as follows:

cellId

Database object ID of the cell to which the pin information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

pinId

Database object ID of the pin to which the information applies

Valid values: Pin ID of any pin in the specified cell

layerNumber

Number of cell layer on which pin is placed

Valid values: Any layer defined in the cell library’s technology file.

License Required

Milkyway

dbSetPinStatusFlag1

Specifies the follow pin information for the pin of a cell.

Syntax

`dbSetPinStatusFlag1 cellId pinName flag`

where the arguments are as follows:

cellId

Database object ID of the cell to which the pin information applies

Valid values:

- Cell ID of any open cell in the current library

- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

pinName

The name of a pin in the cell.

flag

The desired flag value

Valid values:

- “*pinFollowInRow*”
- “*pinFollowInCol*”
- “*pinNoFollow*”

License Required

Milkyway

dbSetPolygonPointArray

Changes the point array in the polygon record. The length of the new point array must match the old length.

Syntax

`dbSetPolygonPointArray cellId polygonId pointArray`

where the arguments are as follows:

cellId

Database object ID of the cell to which the polygon information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

polygonId

Database object ID of the polygon to which the point array information applies

Valid values: Polygon ID of any polygon in the specified cell

pointArray

Coordinates of the polygon

Valid values:

' ((x₁ y₁) , (x₂ y₂) , . . . , (x_n y_n))

License Required

Milkyway

dbSetPortDirection

Specifies the direction of a port.

Syntax

dbSetPortDirection *cellId* "portName" "portDir"

where the arguments are as follows:

cellId

Database object ID of the cell to which port information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

portName

Name of port to which direction specification applies

Valid values: Any port in the specified cell

portDir

Direction that applies to the port

Valid values:

- input
- output
- inputoutput
- tristateoutput
- tristateinputoutput
- none

License Required

Milkyway

dbSetPortInstNetId

Specifies the netId of a port instance without making the port instance a member of the net. Function should be used when connecting power and ground nets to the port instance.

Syntax

`dbSetPortInstNetId cellId portInstId netId`

where the arguments are as follows:

cellId

Database object ID of the cell to which port instance information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

portInstId

Database object ID of the port instance to which net information applies

Valid values: Port instance ID of any port instance in the specified cell

netId

Database object ID of the net

Valid values: Net ID of any net connected to the specified port instance

License Required

Milkyway

dbSetSkipRefLibReadLock

Bypasses the lock on a reference library when you are certain the reference library is read only. No one else can modify the library.

Syntax

```
dbSetSkipRefLibReadLock
```

where the boolean is #t or #f:

#t

Disables ref

#f

Turns the lock back on

License Required

Milkyway

dbSetStdCellOrient

Sets the allowable placement orientations of a standard cell master. This would take effect during the geMove on page 2188 command or the standard cell placement process.

Note:

This command also applies to AstroGA.

Syntax

```
dbSetStdCellOrient <libName> <cellName> <orientation>
```

where the arguments are as follows:

<i>libName</i>	Name of the library containing the cell that the orientation specification applies. Valid values: Name of any library in the current directory.
<i>cellName</i>	Name of the cell that the orientation specification applies. Valid values: <ul style="list-style-type: none">• Name of any cell in the specified library• Pattern matching to select all cells with names that match a pattern. For example, typing <i>TOP.*</i> specifies all cells with names that begin with <i>TOP</i>. For information about pattern matching, see “Specifying Objects by Pattern” in <i>Physical Implementation Getting Started</i>.
<i>orientation</i>	One of {Any Reflect Noreflect} or: combination of {R0 R0_MX R0_MY R180 R90 R90_MX R90_MY R270} Example: "R0 R0_MY R180 R90_MX" These valid values are case insensitive, and spaced by blanks.

Example

```
dbSetStdCellOrient "GOLDDROAD" "DFFRP" "Reflect"
(same as "R0_MX R180")
dbSetStdCellOrient "GOLDDROAD" "INV1" "R0 R0_MY R180"
dbSetStdCellOrient "GOLDDROAD" "NAND3" "R0 R180"
```

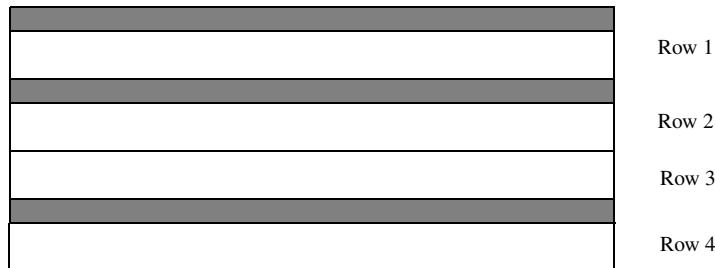
The third example instructs Astro to allow only R0 and R180 orientations during the placement process or the geMove on page 2188 command.

License Required

Milkyway

dbStartCellRowDoubleBack

Specifies the placement orientation of the cell row. When doubleback is set, Astro abuts the cell rows. Rows 2 and 3 in the figure are abutted.



Prerequisites

Prior to executing *dbStartCellRowDoubleBack*, use *dbDeleteCellRowDoubleBack* (see page 3-190) to reset cell row placement.

Syntax

```
dbStartCellRowDoubleBack cellId regionNum startCellRowNo
```

where the arguments are as follows:

cellId

Database object ID of the cell in which cell row information applies

Valid values:

- Cell ID of any open cell in the current library
- (*geGetEditCell*)

For information, see “Specifying Cells” on page 3-57.

regionNum

Number of the cell region in which cell row information applies

Valid values: Positive integers that identify regions inside the specified cell

startCellRowNo

Number of the cell row where cell abutment is to begin

Valid values: Positive integers that identify cell rows inside the specified region

License Required

Milkyway

dbUpdateCellProperty

Updates the value of the specified property of the cell. The standard cell should be opened in .FRAM view.

Syntax

```
dbUpdateCellProperty cellId "propName" value
```

where the arguments are as follows:

cellId

Database object ID of the cell to which information applies

Valid values:

- Cell ID of any open cell in the current library

- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

propName

Name of the cell property to be updated in the specified cell

Valid values: Name of any property

value

Value to which you want Astro to update the cell property

Valid data types:

- string
- float
- integer
- boolean

Note:

If *propName* exists, the type of the value must match the type of the property. If property does not exist, the value can be of any type and the property with the specified value will be created. In the case of a string property, the length of the new string must match the length of the old string.

License Required

Milkyway

dbUpdateCellRailProperty

Updates the y coordinate of an edge of a power/ground rail in a standard cell.

Syntax

`dbUpdateCellRailProperty cellId railSpec side yCoord`

where the arguments are as follows:

cellId

Database object ID of the standard cell to which the y coordinate applies

Valid values:

- Cell ID of any open cell in the current library
- (`geGetEditCell`)

For information, see “Specifying Cells” on page 3-57.

railSpec

Specification of the rail to which the y coordinate applies

Valid values:

- A string that is the name of the pin (rail)
- An integer that is the number of the rail from the bottom up, starting with 1.

side

Side of the rail to which the y coordinate applies

Valid values:

- “low” indicates the bottom side of the rail
- “high” indicates the top side of the rail

yCoord

Y coordinate of the rail edge in database units

Related Command

`dbDisplayCellRailProperty` (see page 3-201) displays the y coordinates of the edges of the power/ground rails in a cell.

License Required

Milkyway

dbRecursiveUpdateAllHierarchy

This command operates in cell hierarchy and propagates any changes to top level cells. It only updates cells that have time stamp's later than the top cell. It is similar to `dbUpdateHierarchy` on page 3-293, except `dbUpdateHierarchy` only works on the top cell's immediate children.

Syntax

`dbRecursiveUpdateAllHierarchy cellId`

where the `cellId` is the database object ID of the cell for which you want to update. The argument `cellId` can be either of the following;

- Cell ID of any open cell in the current library
- (`geGetEditCell`)

For information, see “Specifying Cells” on page 3-57.

Example

Cell A has an instance B, and the master cell of B has an instance C. If the master cell in C has changed, you must do `dbRecursiveUpdateAllHierarchy` to propagate the change to the top level for cells that have a time stamp later than the top cell.

License Required

Milkyway

dbUpdateHierarchy

Compares the time stamps of the parent cell and cell instances updating the cell instances if necessary.

Syntax

`dbUpdateHierarchy cellId`

where the arguments are as follows:

cellId

Database object ID of the parent cell

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

License Required

Milkyway

dbUpdateNetStatus

Updates the status of all nets. Execute this function to correct the status of the net single port connection status.

Syntax

`dbUpdateNetStatus cellId`

where the arguments are as follows:

cellId

Database object ID of the cell to which net information applies

Valid values:

- Cell ID of any open cell in the current library
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

License Required

Milkyway

dbUpdateProperty

Updates the value of the specified property of the specified object in the cell.

Syntax

`dbUpdateProperty cellId objectId "propName" value`

where the arguments are as follows:

cellId

Database object ID of the cell in which information is to be updated

Valid values:

- Cell ID of any open cell in the current library
- (`geGetEditCell`)

For information, see “Specifying Cells” on page 3-57.

objectId

Database object ID of the object to be updated

Valid values:

- *cellInstId*
- *netId*
- *pinId*
- *portId*
- *portInstId*
- *regionPinId*
- *regionPortId*

Note:

This is a partial listing of possible values.

propName

Name of the object’s property to which information applies

value

Value to which you want Astro to update the object property

Valid types:

- string
- float

- integer
- boolean

Note:

If property exists, the type of the value must match the type of the property. If property does not exist, the value can be of any type and the property with the specified value will be created. In the case of a string property, the length of the new string must be the same as the length of the old property.

License Required

Milkyway

dbVerifyCellEEQClass

Verifies the consistency of the content in *EEQClass* of the library.

Syntax

`dbVerifyCellEEQClass libId`

where the arguments are as follows:

libId

Database object ID that must be a library ID.

License Required

Milkyway

dbVerifyCellLEQClass

Verifies the consistency of the content in *LEQClass* of the library.

Syntax

```
dbVerifyCellLEQClass libId
```

where the arguments are as follows:

libId

Database object ID that must be a library ID.

License Required

Milkyway

Extensions for Modifying the Database - Timing

This section provides syntax information for extensions that provide the ability to modify and/or create timing constraints within the Synopsys databases.

dbCreateSGEdge

Creates a slack graph edge record.

Syntax

```
dbCreateSGEdge cellId status fromObjId toObjId
```

where the arguments are as follows:

cellId

Database object ID of the cell to which timing information applies

Valid values: Cell ID of any cell in the current library

status

Valid values:

- 0

fromObjId

Database object ID of the object from which timing signal leaves

Valid values: Object ID of any object in the current cell

toObjId

Database object ID of the object to which timing signal comes

Valid values: Object ID of any object in the current cell

License Required

Milkyway

dbCreateSGPrimaryNode

Creates a slack graph primary node record in a specified cell

Syntax

```
dbCreateSGPrimaryNode cellId status riseMax riseMin fallMax  
fallMin startOrEndId
```

where the arguments are as follows:

cellId

Database object ID of the cell to which timing information applies

Valid values: Cell ID of any cell in the current library

status

Whether slack graph primary node should be an arrival time node or required arrival time node

Valid values:

- 0 means that the node is an arrival time node.
- 1 means that the node is a required arrival time node.

riseMax

Latest time the signal arrives or should arrive on a rising edge

Valid values:

- If status is set to 0, value represents the latest time the signal will arrive.
- If status is set to 1, value represents the latest time the signal should arrive.

riseMin

Earliest time the signal arrives or should arrive on a rising edge

- If status is set to 0, value represents the earliest time the signal will arrive.
- If status is set to 1, value represents the earliest time the signal should arrive.

fallMax

Latest time the signal arrives or should arrive on a falling edge

- If status is set to 0, value represents the latest time the signal will arrive.
- If status is set to 1, value represents the latest time the signal should arrive.

fallMin

Earliest time the signal arrives or should arrive on a falling edge

- If status is set to 0, value represents the earliest time the signal will arrive.
- If status is set to 1, value represents the earliest time the signal should arrive.

startOrEndId

Database object ID of the object at either the start or end of the timing path

Valid values: Object ID of any object in the specified cell

License Required

Milkyway

dbSetCellPowerValue

Sets the power value of a specified cell.

Prerequisites

The units for power must be specified in the unit section of your technology file. For example,

```
*unit  
power mw 1000  
*end
```

For more information, see “Power” on page 12 of the *Milkyway Data Preparation User Guide*.

Syntax

```
dbSetCellPowerValue cellId powerValue
```

where the arguments are as follows:

cellId

ID of the cell to which power value applies

Valid values: ID of any open cell in the current library

powerValue

Amount of power the cell consumes

Valid types:

- Float

- Integer

License Required

Milkyway

dbSetFringeFactor

Specifies the fringe factor between two objects.

Syntax

```
dbSetFringeFactor layerNo1 layerNo2 minFringeFactor  
                  maxFringeFactor
```

where the arguments are as follows:

layerNo1

Number of layer in which first object is located

Valid values: Any layer in the current library

layerNo2

Number of layer in which second object is located

Valid values: Any layer in the current library except the layer specified in *layerNo1*

minFringeFactor

Minimum value for fringe factor

Valid values: A positive or negative integer

maxFringeFactor

Maximum value for fringe factor

Valid values: A positive or negative integer

License Required

Milkyway

dbSetLayerTimeInfo

Specifies the following timing information for a layer:

- Resistance
- Capacitance
- Inductance

Syntax

```
dbSetLayerTimeInfo layerNumber minRes maxRes minCap maxCap
minInd maxInd
```

where the arguments are as follows:

layerNumber

Number identifying the layer to be modified

Valid values: A positive integer that corresponds to an existing layer

minRes

Value that specifies the minimum resistance allowable for the layer specified by *layerNumber*

maxRes

Value that specifies the maximum resistance allowable for the layer specified by *layerNumber*

minCap

Value that specifies the minimum capacitance allowable for the layer specified by *layerNumber*

maxCap

Value that specifies the maximum capacitance allowable for the layer specified by *layerNumber*

minInd

Value that specifies the minimum inductance allowable for the layer specified by *layerNumber*

maxInd

Value that specifies the maximum inductance allowable for the layer specified by *layerNumber*

License Required

Milkyway

dbSetLModelSubType

Specifies the logic model subtype of a cell, including any latch, flip-flop, RAM, ROM, or clock cell in the library.

Caution!

Because *dbSetLModelSubType* functions set the types on the timing (.tim) cells, you need to be sure to create these timing cells before you load the *dbSetLModelSubType* functions. You create timing cells by loading *defineTimeIntrinsic* and *definePortCapacitance* functions.

Note:

You can include *dbSetLModelSubType* functions in a CLF file, along with the other functions used to define the timing library for slack graph generation (*defineTimeIntrinsic*, *definePortCapacitance*, and *dbSetCellPortTypes* (see page

3-257). However, you need to place the defineTimeIntrinsic (see page 3449 of the *IC Implementation Command Reference Manual*) and definePortCapacitance (see page 3430 of the *IC Implementation Command Reference Manual*) functions before the dbSetLModelSubType functions in the file, so that the timing cells exist when the dbSetLModelSubType functions attempt to set the types.

Syntax

```
dbSetLModelSubType cellId "LModelSubType"
```

where the arguments are as follows:

cellId

ID of the timing (.tim) cell for which you are specifying a type
Valid values: ID of any timing cell in the current directory. If you do not know the cell ID, see “Specifying Cell IDs” on page 1-6 of the *Static Timing Analyzer User Guide* .

type

Type of the cell

Valid values: Any of the following:

- *latch*
- *flipflop*
- *ram*
- *rom*
- *clock* (clock buffer definition for clock tree synthesis)

Example

```
setq cellId (dbOpenCell "latch1.tim" "w")
dbSetLModelSubType cellId "latch"
dbCloseCell cellId
```

dbSetLModelSubType

4-267

These statements do the following:

- The first line opens the cell *latch1.tim* and creates the variable *cellId* for its cell ID.
- The second line specifies that the cell is a latch.
- The third line closes the cell.

Related Commands

“Matrices for Timing Functions” on page A-1 of the *Static Timing Analyzer User Guide* contains an annotated list of timing functions.

License Required

Milkyway

dbSetPathConstraint

This timing command sets the minimum and maximum rise and fall delays for a path. In other words, this function sets *riseMin*, *riseMax*, *fallMin*, and *fallMax* values to specify that the signal propagating through a defined path should arrive

- No sooner than
 - *riseMin* for a rising signal at the end point of the path
 - *fallMin* for a falling signal at the end point of the path
- No later than
 - *riseMax* for a rising signal at the end point of the path
 - *fallMax* for a falling signal at the end point of the path

Syntax

```
dbSetPathConstraint riseMin riseMax fallMin fallMax  
"startPoint" "endPoint" ['({{"interimPoint"} ...})]
```

where the arguments are as follows:

riseMin

Minimum delay of the transition of the input to the rise of the output

Valid values: Any floating-point number less than *riseMax*

riseMax

Maximum delay of the transition of the input to the rise of the output

Valid values: Any floating-point number greater than *riseMin*

fallMin

Minimum delay of the transition of the input to the fall of the output

Valid values: Any floating-point number less than *fallMax*

fallMax

Maximum delay of the transition of the input to the fall of the output

Valid values: Any floating-point number greater than *fallMin*

startPoint

Name of the first pin along the path

Valid values: If the pin is a top-level pin, specify the name of the pin. If the pin is on a cell instance, specify the name of the pin in the following format:

pinName/*instanceName*

where *pinName* is the name of the pin and *instanceName* is the name of the cell instance.

endPoint

Name of the last pin along the path

Valid values: Same as *startPoint*

interimPoint

Name of an interim pin along the path

Valid values: Same as *startPoint*. When specifying multiple pins, specify them in proper order and separate the names with spaces.

Note:

The square brackets ([]) indicate that this argument is optional.

If you are setting a constraint between two pins with no interim pins, you do not need to specify this argument.

Example 1

```
dbSetPathConstraint 2.0 3.0 2.1 3.1 "P1/I255" "P20" ' ("P1  
I133" "P2/I115")
```

This function sets a constraint for a path from pin *P1* on cell instance *I255* to pin *P1* on cell instance *I133* to pin *P2* on cell instance *I115* to the top-level pin *P20*. During timing-driven placement, Astro attempts to place cells such that a signal propagating through the path arrives

- Rising no sooner than 2.0 ns and no later than 3.0 ns
- Falling no sooner than 2.1 ns and no later than 3.1 ns

Example 2

```
dbSetPathConstraint 2.0 3.2 1.9 3.1 "P4/I135" "P12"
```

This function sets a constraint for a direct path from pin *P4* on cell instance *I135* to the top-level pin *P12*. During timing-driven placement, Astro attempts to place cells such that a signal propagating through the path arrives

- Rising no sooner than 2.0 ns and no later than 3.2 ns
- Falling no sooner than 1.9 ns and no later than 3.1 ns

Related Commands

“Matrices for Timing Functions” on page A-1 of the *Static Timing Analyzer User Guide* contains an annotated list of timing functions.

License Required

Milkyway

AstroGA Database Extensions

This section provides syntax information for extensions that provide the ability to interface with the AstroGA database.

Note:

Only AstroGA uses the commands presented in this section.

dbSetCellPattern

Defines the legal orientations of the cell when placed on each of the tiles in the repeating pattern of a tile array. The tiles are referred to by their positions as “LL” (lower left), “LR” (lower right), “UL” (upper left), and “UR” (upper right).

Repeating pattern →

UL	UR	UL	UR	UL	UR
LL	LR	LL	LR	LL	LR
UL	UR	UL	UR	UL	UR
LL	LR	LL	LR	LL	LR

This function returns #t when successful.

Syntax

```
dbSetCellPattern cellId patternNumber1 patternNumber2  

patternNumber3 patternNumber4
```

where the arguments are as follows:

cellId

Database object ID that must be a cell ID.

Valid values:

- Cell ID of any cell in the open library.
- (geGetEditCell)

For information, see “Specifying Cells” on page 3-57.

patternNumber1

Legal orientation for the lower left tile

Valid values: Any combination of the following:

- If bit0 is set, AstroGA places the cell with no transformation.
- If bit1 is set, AstroGA places the cell flipped on the Y axis.
- If bit2 is set, AstroGA places the cell flipped on the X axis.

- If bit3 is set, AstroGA places the cell rotated 180 degrees.
- If bit4 is set, AstroGA places the cell flipped on the X axis and rotated 90 degrees.
- If bit5 is set, AstroGA places the cell rotated 90 degrees.
- If bit6 is set, AstroGA places the cell rotated 270 degrees.
- If bit7 is set, AstroGA places the cell flipped on the Y axis and rotated 90 degrees.

patternNumber2

Legal orientation for the lower right tile.

Valid values: See the valid values from *patternNumber1* above.

patternNumber3

Legal orientation for the upper left tile.

Valid values: See the valid values from *patternNumber1* above.

patternNumber4

Legal orientation for the upper right tile.

Valid values: See the valid values from *patternNumber1* above.

License Required

Milkyway

dbAddCellInstToPlanGroupByName

Adds a cell instance to plan group.

Syntax

```
dbAddCellInstToPlanGroupByName cellId pgId "ciName"
```

dbAddCellInstToPlanGroupByName

where the arguments are as follows:

cellId

Database object ID of the cell to which the information applies.

Valid values:

* Cell ID of any open cell in the current library.

* (geGetEditCell).

For information, see Specifying Cells on page 3-57.

pgId

Database object ID of the plan group in the cell.

Valid values:

planGroup: Id of any plan group in the cell.

ciName:

Name of the *cellInst* in the cell that you want to add to the plan group.

License Required

Milkyway

dbAddNetSubTypeByName

Add new net subType to a specific Net in the cell.

Syntax

```
dbAddNetSubTypeByName cellId "netName" "netType"
```

where the arguments are as follows:

cellId

Database object ID of the cell to which the information applies.

Valid values:

* Cell ID of any open cell in the current library.

* (geGetEditCell)

For information, see Specifying Cells on page 3-57.

netName

Name of the net that you want to add the type.

Valid values: Name of any net in the open cell.

netType

Type you want to add to net.

Valid values: "core", "pad", "coreAndPad", "CTSNet",
"CTSRootNet", "nonCTSNet"

License Required

Milkyway

dbAddPartialScanToScanChain

Adds specified partial scan cell Instances to a scan chain

Syntax

```
dbAddPartialScanToScanChain cellId "chainName"  
' ({lItem}...)
```

where the arguments are as follows:

cellId

Database object ID of the cell to which the information applies.

Valid values:

* Cell ID of any open cell in the current library.

* (*geGetEditCell*)

For information, see Specifying Cells on page 3-57.

`chainName`

Name of the scan chain to which you are adding a partial scan.

Valid values: Name of any scan chain in the open cell.

`lItem`

List of the `cellInst` name that you want in the group.

Valid values: Name of any `cellInst` in the open cell.

License Required

Milkyway

dbArchiveCellToLib

This function archives a cell design to your specified archive library. All design data (including unit tiles, library cell equivalence information, and library global port tables) are copied into a single library. The archive library will not contain any hierarchical reference information.

The versions, views, and reference cells that are archived may be controlled by optional arguments. When the default settings are used, all versions, views, and reference cells of the cell design are archived.

When `dbArchiveCellToLib` completes successfully, a `true` message is returned. When unsuccessful, a `false` message is returned.

Prerequisites

The library containing the cell to be archived must be open. If the library uses reference control files, the reference control file mode must be set with dbSetRefControl FileMode (see page 1736 of the *IC Implementation Command Reference Manual*) prior to opening the library. Otherwise, cells that are referenced by the top cell and are located in reference libraries will not be copied.

Syntax

```
dbArchiveCellToLib cellName archiveLibName  
[versionMode topCellOnly overWrite] [userDefinedProp]  
[excludeViewsProp] [includeViewsProp] [allViewCellsProp]  
[warningLevelProp] [skipDupCellsProp]
```

where the arguments are as follows:

cellName

The name of the source top cell.

Valid Values:

Syntax:

```
sourceCellName[;sourceVersion]  
[=newCellName]
```

- “;” - Separates *sourceCellName* from the *sourceVersion*.
- “=” - Separates the *sourceCellName* from the *newCellName*.
- *sourceCellName* - Name of the source top cell to be archived.
- *sourceVersion* - Version of the source top cell to archive (optional)
- *newCellName* - New cell name to which the source top cell will be copied in the archive library (optional).

Examples: If the value is “*top;3*”, the third version of the cell “*top*” will be archived. If the value is “*top=newTop*”, the cell “*top*” will be archived as “*newTop*”. If the value is “*top;3=newTop*”, the third version of the cell “*top*” will be archived as “*newTop*”.

<i>archiveLibName</i>	The name of the archive library. Valid Values: Any existing or non-existing library. If the library does not exist, it will be created using the same technology information as the source library.
<i>versionMode</i>	Specifies which cell versions are archived (optional). Note: If a particular version is specified in the cellName (see above) for the top cell, it will only take effect if versionMode has a 1 or 2 value.
<i>topCellOnly</i>	Valid Values: <ul style="list-style-type: none"> • 0: Archives all versions (default). • 1: Archives the most recent version only. • 2: Archives the version used to create the particular cell instance only. Specifies whether to archive all reference cells or only the top cell (optional). Valid Values: <ul style="list-style-type: none"> • 0: Archives the top cell and all reference cells (default). • 1: Archives only the top cell.
<i>overwrite</i>	Specifies if existing archived cells should be overwritten (optional). Valid Values: <ul style="list-style-type: none"> • 0: Existing cells will not be overwritten, instead new versions will be created in the archive library (default). • 1: The most current version of the existing cells will be overwritten. If the cell does not exist in the archive, a new version will be created.

userDefinedProp

A property string that describes a user defined property to be stored with the archived cells (optional). This property is stored with all archived cell views and versions.

Valid Values:

Syntax: [propName=]propValue

- “=” - Separates the property name and property value.
- *propName* - The name of the property (optional). If not specified, the default property name will be “DB_ARCHIVE_PROP”. If the property name already exists, it will be updated if the property value is different.
- *propValue* - The string value of the property

Note:

“EXCLUDE_VIEWS” and “INCLUDE_VIEWS” are key property names. They cannot be used as property names in userArchProp.

excludeViewsProp

A property string that describes the views to be excluded from the archive library (optional). This property is not stored with the archived cells.

Valid Values:

Syntax: EXCLUDE_VIEWS=propValue

- “=” - Separates the property name and property value.
- “EXCLUDE_VIEWS” - A key property name, and may not be used as a user archive property (see above).
- *propValue* - A comma (and/or space) delimited list of cell views that should not be archived. All other cell views will be included in the archive.

includeViewsProp

A property string that describes the views to be included in the archive library (optional). This property is not stored with the archived cells.

Valid Values:

Syntax: INCLUDE_VIEWS=propValue

- “=” - Separates the property name and property value.
- “INCLUDE_VIEWS” - A key property name, and may not be used as a user archive property (see above).
- *propValue* - A comma (and/or space) delimited list of cell views that should be archived. All other unlisted views will be excluded from the archive.

Note:

The “EXCLUDE_VIEWS” and “INCLUDE_VIEWS” properties are mutually exclusive. They cannot be used in the same function call.

allViewCellsProp xxxxx

warningLevelProp xxxxx

skipDupCellsProp This argument can either be
 “SKIP_DUP_CELLS=TRUE” or
 “SKIP_DUP_CELLS=FALSE”. If TRUE, then only
 the first copy of each cell is archived. Duplicated
 cells are not archived. By default, the value is FALSE.

Related Commands

[dbArchiveHierDesign](#) on page 3-323 - archives a cell design maintaining the original library referencing structure.

License Required

Milkyway

dbArchiveHierDesign

Archives the specified cell design data to a specified location. In order to meet the requirements of portability and hierarchical referencing, *dbArchiveHierDesign* will maintain the original library directory structure under the archive location. All necessary data for the main cells referenced in the top cell are copied over to the archive location. The library references maintained in the archive libraries are updated to reflect the new locations of the reference libraries in the archive location.

If *dbArchiveHierDesign* completes successfully, a *true* message is returned. If unsuccessful, a *false* message is returned.

Prerequisites

All libraries must be closed

Syntax

```
dbArchiveHierDesign "libName" "cellName"  
"archiveDirectory"
```

libName

Path (relative or absolute) and name of the library containing the cell to be archived.

Valid values: Any existing library

cellName

Cell name of the cell to be archived.

archiveDirectory

Location to which the cell will be archived.

Valid values: Any existing or nonexisting directory. If the directory already exists, the contents maybe overwritten.

Related Commands

`dbRestoreHierDesign` on page 3-351 - restores a design that was archived using `dbArchiveHierDesign`

`dbArchiveCellToLib` on page 3-317 - archives a cell design into a single library

License Required

Milkyway

dbClearMustJoinPortByCell

Clears all ports in a cell.

Syntax

```
dbClearMustJoinPortByCell "cellName"
```

where the arguments are as follows:

cellName

Name of the cell that you want to clear the must join port.

Valid values: name of any cell in the library.

License Required

Milkyway

dbCreateBaseArray

Creates *BaseArray* object.

Syntax

```
dbCreateBaseArray cellId '({llPointArray}...) "className"  
tileWidth tileHeight
```

where the arguments are as follows:

cellId

Database object ID of the cell to which the information applies.

Valid values:

* Cell ID of any open cell in the current library.

* (*geGetEditCell*)

For information, see Specifying Cells on page 3-57.

llPointArray

Coordinates indicating the size and shape of the *BaseArray*.

Valid syntax:

$((x_1 \ y_1) \ (x_2 \ y_2) \ ... \ (x_1 \ y_1))$

className

Name of the class which the *BaseArray* applies.

Valid values: Name of any class in the open cell.

tileWidth

The tile dimension of the width.

Valid values: any positive integer

tileHeight

The tile dimension of the height.

Valid values: any positive integer

License Required

Milkyway

dbCreateBaseArrayRow

Create a row in the *baseArray*.

Syntax

```
dbCreateBaseArrayRow cellId baseArrayId PointArray
```

where the arguments are as follows:

cellId

Database object ID of the cell to which the information applies.

Valid values:

* Cell ID of any open cell in the current library.

* *(geGetEditCell)*

For information, see Specifying Cells on page 3-57.

baseArrayId

Database object ID of the *baseArray* in the open cell

Valid values: *baseArrayId* of the *baseArray* object in the open cell.

PointArray

Coordinates indicating the size and shape of the *BaseArray*.

Valid syntax:

'((x1 y1) (x2 y2))

License Required

Milkyway

axCreateContactArray

Creates (or adds) contact instances arrays into a design. Contacts are predefined in your technology file. See *ContactCode* Section for more information.

Syntax

```
axCreateContactArray cellId "NetName" contactNumber  
transform  
flag Point xPitch yPitch xTimes yTimes BitMap
```

where the arguments are as follows:

cellId

Database object ID of the cell to which the information applies.

Valid values:

* Cell ID of any open cell in the current library.

* (geGetEditCell)

For information, see Specifying Cells on page 3-57.

NetName

The name of the net on which you want to add the Contact Array.

Valid values: any net in the open cell.

ContactNumber

Number you specified in the *deviceTable* section of the technology file that identifies the contact you want to create.

Valid values: Any integer from 1 to 255 used in the *deviceTable* to identify a contact. For the following *deviceTable* section, the *contactNumber* for the contact, polymet1, is 1.

```
*deviceTable
*contact 1 polymet1
cont poly m1 1 1 1.2 2.5 2.0
defaultRowNumber 2
parasitics 0.2 0.0 0.0
*end
```

transform

Whether or not to rotate all contacts in the array by 90 degrees.

Valid values:

* 0 means the contacts are not rotated

* 1 means the contacts are rotated 90 degrees

flag

The desired flag value

Valid values: Positive integers

Point

Point corresponding to the center of the contact array.

Valid syntax:

(x y)

xPitch

The center-to-center spacing of the contact cuts in the horizontal direction.

Valid Values:

number of units to define the X dimension spacing of your array.

yPitch

The center-to-center spacing of the contact cuts in the vertical direction.

Valid Values:

number of units to define the Y dimension spacing of your array.

xTimes

Number of sites created in the x direction.

yTimes

Number of sites created in the y direction.

bitmap

If the contact array is full, then there is no need to use bitmap, e.g. bitmap is NULL. Otherwise, a row-wise bit representation is used to indicate some contact in the contact array is empty when the bit is 0. An int32 can hold 32 bits. So the size of the bitmap can be determined by:

```
tmp = sizeof (unit32);
```

```
mapSize = (bitMap?((xTimes*yTimes - 1)/(8*tmp)+1)*tmp:0);
```

License Required

Milkyway

dbCreateErrorComponent

Create an error component object and attaches it to an error object. An error component consists of a set of one or more rectangles that graphically highlight the location of an error. The rectangles are supplied as a even numbered list of coordinate pairs supplied to the function as the *pointArray* parameter. Each pair of coordinates represents opposing corners of a rectangle.

Syntax

```
dbCreateErrorComponent cellId errObjId nodeNum "netName"  
pointList
```

where the arguments are:

cellId

Database object ID of the cell to which the information applies.

Valid values:

* Cell ID of any open cell in the current library.

* (*geGetEditCell*)

For information, see Specifying Cells on page 3-57.

errObjId

Database object ID of the error object where the error component information applies. The errorObject can be created using *dbCreateErrorObj*.

Valid values: *errObjId* of any error object in the open cell.

nodeNum

Represents the integer value that is displayed by GUI commands such as *geAllError*.

netName

Name of net to which error component relates. The value is not currently used.

Valid values: Name of any net in the open cell which error component relates.

pointList

Represents the array of points describing 1 or more rectangles.

Valid values:

Must be an even number of points, each pair representing opposite corners of a rectangle.

License Required

Milkyway

dbCreateErrorObj

Creates an error object. An error object represents one error. The error object is associated with a single error type and might have numerous error components. *dbCreateErrorObj* returns the database object ID of the new “ErrorObject.”

Syntax

```
dbCreateErrorObj cellId errTypeID "Info"
```

where the arguments are as follows:

<i>cellId</i>	Database object ID of the cell to which the information applies. Valid values: <ul style="list-style-type: none">• Cell ID of any open cell in the current library: <i>(geGetEditCell)</i>. For information, see Specifying Cells on page 3-57.
<i>errTypeID</i>	Database object ID of the error type where the error object information applies. Valid values: <i>errTypeID</i> of any error Type in the open cell. The <i>errorType</i> can be created using <i>dbCreateErrorType</i> .
<i>Info</i>	Specifies the information related to the error. Valid values: String of char.

License Required

Milkyway

dbCreateErrorType

Create an error type. An error type represents a particular kind of error and might have several related error objects.

dbCreateErrorType returns the database object ID of the new “*errorType*”.

Syntax

```
dbCreateErrorType cellId errorLevel "typeName" "info"
```

where the arguments are:

cellId

Database object ID of the cell to which the information applies.

Valid values:

* Cell ID of any open cell in the current library.

* (geGetEditCell).

For information, see Specifying Cells on page 3-57.

errorLevel

Specifies the severity of the error type. The value is not used now.

typeName

Name of the *errorType*. The length is limited to 15 characters.

Valid values:

Name of the creating *errorType*.

Info

Specifies the information related to the error.

Valid values: String of char.

License Required

Milkyway

dbCreatePath

Create path record for the cell in the database

Syntax

```
dbCreatePath cellId "layerName" pathType width routeType  
dataType PointArray
```

where the arguments are:

cellId

Database object ID of the cell to which the information applies.

Valid values:

- Cell ID of any open cell in the current library
- (*geGetEditCell*)

For information, see Specifying Cells on page 3-57.

layerName

Name of layer to which path information applies.

Valid values:

Name of any layer in the specified cell.

pathType

The type of created path

Valid values:

- 0 when the path edges extend with the coordinate.
- 1 when the path edge is rounded.
- 2 when the path edge extends a half width beyond the coordinate.

width

The desired width for the creating path.

routeType

routeType of created path

Valid values:

Only bit (1-5) are used in the routing type definition.

Major type - 2 bits

- 0 axcSignalRouteType
- 16 axcPGRouteType
- 32 axcClockRouteType
- 48 axcMiscRouteType

dataType

DataType to create the path

pointArray

Coordinates indicating the size and shape of the path

Valid syntax: '((x1 y1) (x2 y2)....)

License Required

Milkyway

axCreateRowRecord

Create Row record in the database

Syntax

```
axCreateRowRecord cellId direction flip "tilePatternName"  
listLL  
listUR
```

where the arguments are:

cellId

Database object ID of the cell to which the information applies.

Valid values:

* Cell ID of any open cell in the current library.

* (geGetEditCell)

For information, see Specifying Cells on page 3-57.

direction

Preferred direction.

Valid values: vertical = 1 and horizontal = 0

flip

Specifies whether the tile will be flipped or not. 0 means not flipped.

tilePatternName

TilePattern name in the row.

Valid values: any tile pattern name from the open cell that you can get the tile pattern number from the name.

listLL

Coordinate that indicates the Left segment extends Low of the bbox.

Valid syntax: '(x y)

listUR

Coordinate that indicates the upper right of the bbox.

Valid syntax: '(x y)

License Required

Milkyway

axCreateTilePatternRecord

Create a *tilePattern* record in the open cell.

Syntax

```
axCreateTilePatternRecord cellId "tileName" "patternName"  
    direction flip
```

where the arguments are:

cellId

Database object ID of the cell to which the information applies.

Valid values:

* Cell ID of any open cell in the current library.

* (geGetEditCell)

For information, see Specifying Cells on page 3-57.

tileName

tile name in the open cell.

If not specified, i.e. == NULL, then the default “unit” will be used.

patternName

pattern name in the row.

Valid values: any pattern name from the open cell that you can get the tile pattern number from the name.

direction

Preferred direction.

Valid values: vertical = 1 and horizontal = 0

flip

Specifies whether the tile will be flipped or not. 0 means not flipped.

License Required

Milkyway

axCreateTrackRecord

Create a track record in the open cell.

Syntax

```
axCreateTrackRecord cellId layerNumber direction startTrack  
step  
  xxtimes listLL listUR
```

where the arguments are as follows:

<i>cellId</i>	Database object ID of the cell to which the information applies. Valid values: <ul style="list-style-type: none">• Cell ID of any open cell in the current library: <i>(geGetEditCell)</i>. For information, see Specifying Cells on page 3-57.
<i>layerNumber</i>	Number of cell layer to which rectangle information applies. Valid values: positive integer (0 - 256)
<i>direction</i>	Preferred direction. Valid values: vertical = #f, horizontal = #t
<i>startTrack</i>	Number of the start track.
<i>step</i>	The distance between each track.
<i>xxtimes</i>	Number of tracks created.
<i>listLL</i>	Coordinate that indicates the Left segment extends Low of the bbox. Valid syntax: '(x y)
<i>listUR</i>	Coordinate that indicates the upper right of the bbox. Valid syntax: '(x y)

Example

```
axCreateTrackRecord _cell 2 #t 0.32 0.64 15 '(0.0000 0.3200)
  '(0.6400 9.2800)
```

License Required

Milkyway

axCreateWireDirRecord

The command will create a wire direction record in the open cell.

Syntax

```
axCreateWireDirRecord cellId layer direction
```

where the arguments are as follows:

cellId Database object ID of the cell to which the information applies.

Valid values:

- Cell ID of any open cell in the current library:
(*geGetEditCell*).

For information, see Specifying Cells on page 3-57.

layer Number of cell layers to which the rectangle information applies.

Valid values:

positive integers (0 - 256)

direction Preferred direction.

Valid values:

vertical = #f, horizontal = #t

Example

```
axCreateWireDirRecord _cell 32 #t
```

License Required

Milkyway

geOutputPoint

Output the coordinate of the point from the mouse click.

Syntax

geOutputPoint

License Required

Milkyway

geOutputLine

Output the coordinate of the line from the mouse click.

Syntax

geOutputLine

License Required

Milkyway

geOutputBBox

Output the coordinate of the bbox from the mouse click.

Syntax

geOutputBBox

License Required

Milkyway

dbPurgeAllPlanGroup

Purge all plan group in the open cell.

Syntax

```
dbPurgeAllPlanGroup cellId
```

where the arguments are:

cellId

Database object ID of the cell to which the information applies.

Valid values:

- * Cell ID of any open cell in the current library.
- * (geGetEditCell).

For information, see Specifying Cells on page 3-57.

Licences Required

Milkyway

dbRemoveInstGroupFromScanChain

Removes *InstGroup* from scan chain in the open cell.

Syntax

```
dbRemoveInstGroupFromScanChain cellId "chainName"  
"instGroupName"
```

where the arguments are:

cellId

Database object ID of the cell to which the information applies.

Valid values:

* Cell ID of any open cell in the current library.

* (geGetEditCell).

For information, see Specifying Cells on page 3-57.

chainName

Name of the scan chain that you want to remove inst group from.

Valid values: Any scan chain name that contains in the inst group in the open cell.

instGroupName

Name of the *instGroup* that you want to remove from the scan chain.

Valid values: Any valid name of the *instGroup* in the open cell.

License Required

Milkyway

dbReplaceCellTextChar

Replace cell text character in the open cell.

Syntax

```
dbReplaceCellTextChar cellId "oldCharSet" "newChar"
```

where the arguments are:

cellId

Database object ID of the cell to which the information applies.

Valid values:

* Cell ID of any open cell in the current library

* (geGetEditCell)

For information, see Specifying Cells on page 3-57.

oldCharSet

Specifies the set of characters to change

newChar

The character to which all characters in *oldCharSet* will be changed.

Note:

If newChar includes more than one char, then only the first char in the string will be used for the replacement.

License Required

Milkyway.

dbReplaceCellTextStr

Replace cell text string in the open cell.

Syntax

```
dbReplaceCellTextStr cellId "oldText" "newText"
```

where the arguments are:

cellId

Database object ID of the cell to which the information applies.

Valid values:

* Cell ID of any open cell in the current library.

* (geGetEditCell).

For information, see Specifying Cells on page 3-57.

oldCharText

Specifies the text objects to change.

newText

Specifies the new value to give to text objects being changed.

License Required

Milkyway.

dbReplaceChildCell

Replace all cell instances of the old child cell with the new child cell name.

Syntax

```
dbReplaceChildCell cellId "oldCellName" "oldViewName"  
"newCellName" "newViewName" "topCellName"
```

where the arguments are:

cellId

Database object ID of the cell to which the information applies.

Valid values:

* Cell ID of any open cell in the current library.

* (geGetEditCell)

For information, see Specifying Cells on page 3-57.

oldCellName

Name of the old child cell in the open cell that you want to replace with the new child cell name.

`oldViewName`

View name of the old child cell in the open cell that you want to replace with the new child cell view name.

`newCellName`

Name of the new child cell that you want to replace for the old one.

`newViewName`

View name of the new child cell that you want to replace for the old one.

`topCellName`

Name of the top cell in the open cell.

License Required

Milkyway

dbRestoreHierDesign

Restores the specified archived cell to the current directory. The cell design must have been archived with the dbArchiveHierDesign function. To meet the requirements of portability and hierarchical referencing, dbRestoreHierDesign will restore the original library directory structure to the current directory. All necessary data for the main and reference libraries will be copied over to the current directory. All cells referenced in the top cell are copied over to the current directory. The library references maintained in the restored libraries are updated to reflect the new locations of the reference libraries in the current directory.

If dbRestoreHierDesign completes successfully, a *true* message is returned. If unsuccessful, a *false* message is returned.

Prerequisites

All libraries must be closed.

Syntax

```
dbRestoreHierDesign "archiveLibName" "cellName"
```

where the arguments are:

archiveLibName

Path (relative or absolute) and name of the archived library containing the cell to be restored.

Valid values:

Any library that was created using dbArchiveHierDesign.

cellName

Specifies the text objects to change.

Related Commands

dbArchiveHierDesign (see page 3-323) - archives a cell design maintaining the original library referencing structure.

dbArchiveCellToLib on page 3-317 - archives a cell design into a single library.

License Required

Milkyway.

dbSetCellInstPlacement

Set *CellInst* Placement in the open cell.

Syntax

```
dbSetCellInstPlacement cellId "cellInstName" rotationStr  
mirrorStr "justStr" Point
```

where the arguments are:

cellId

Database object ID of the cell to which the information applies.

Valid values:

* Cell ID of any open cell in the current library.

* (geGetEditCell)

For information, see Specifying Cells on page 3-57.

cellInstName

Name of the *cellInst* in the open cell to which the placement changes apply.

rotationStr

Specifies the counter-clockwise rotation value in degrees (0, 90, 180, or 270).

mirrorStr

Specifies the direction if any in which to flip cell instance (no, X, or Y).

justStr

Defines the placement of the cell instance around the origin.

Valid values:

- * ll - places the cell instance to the lower left of the origin.
- * lc - places the cell instance to the lower center of the origin.
- * lr - places the cell instance to the lower right of the origin.
- * cl - places the cell instance to the center left of the origin.
- * cc - centers the cell instance around the origin.
- * cr - places the cell instance to the center right of the origin.
- * ul - places the cell instance to the upper left of the origin.
- * uc - places the cell instance to the upper center of the origin.
- * ur - places the cell instance to the upper right of the origin.

Point

Specifies the x-y coordinate pair representing the location of the cell instance origin. Valid syntax: '(x y).

License Required

Milkyway

dbSetCellMustJoin2Pin

Instructs the Synopsys application to externally connect together specified pins on a port.

Syntax

```
dbSetCellMustJoin2Pin cellId "portName" "pinName1"  
"pinName2"
```

where the arguments are:

cellId

Database object ID of the cell to which the information applies.

Valid values:

* Cell ID of any open cell in the current library.

* (geGetEditCell)

For information, see Specifying Cells on page 3-57.

`portName`

Name of the port for which you want to specify external pin connections.

`pinName1`

Names of the first pin you want the application to connect externally.

`pinName2`

Names of the second pin you want the application to connect externally.

License Required

Milkyway

dbSetCellMustJoinPinPattern

Establishes external connection specification for all pins for a port uniquely identified by given *portNamePattern*.

Syntax

```
dbSetCellMustJoinPinPattern cellId "portNamePattern"
```

where the arguments are:

`cellId`

Database object ID of the cell to which the information applies.

Valid values:

* Cell ID of any open cell in the current library.

* (geGetEditCell)

For information, see Specifying Cells on page 3-57.

portNamePattern

Specifies the pattern that uniquely identifies a port in the cell.

License Required

Milkyway

dbSetErrorTypeStatus

Sets the value of the status field of an error type. The error status is not currently used by Synopsys tools.

Syntax

```
dbSetErrorTypeStatus cellId ErrorTypeId status
```

where the arguments are:

cellId

Database object ID of the cell to which the information applies.

Valid values:

* Cell ID of any open cell in the current library.

* (geGetEditCell)

For information, see Specifying Cells on page 3-57.

errorTypeld

Specifies the database identifier of error type being updated.

Valid values: Any *errorTypeld* in the open cell.

status

Specifies the value to assign to status field of error type.

License Required

Milkyway

dbSetErrorObjStatus

Sets the value of the status field of an *errorObj*. The error status is not currently used by Synopsys tools.

Syntax

```
dbSetErrorObjStatus cellId ErrorObjId status
```

where the arguments are:

cellId

Database object ID of the cell to which the information applies.

Valid values:

* Cell ID of any open cell in the current library.

* (geGetEditCell)

For information, see Specifying Cells on page 3-57.

errorObjId

Specifies the database identifier of *errorObj* being updated.

Valid values: Any *errorObjId* in the open cell.

status

Specifies the value to assign to status field of error type.

License Required

Milkyway

dbSetNetSubTypeByName

Sets a sub-type for the net specified by its name.

Syntax

```
dbSetNetSubTypeByName cellId "netName" "netType"
```

where the arguments are:

cellId

Database object ID of the cell to which the information applies.

Valid values:

- * Cell ID of any open cell in the current library.
- * (geGetEditCell)

For information, see Specifying Cells on page 3-57.

netName

Names of any net in the open cell.

netType

Specifies the net sub-type.

Valid values:

- * core
- * pad
- * *coreAndPad*
- * *CTSNet*
- * *CTSRootNet*

* *nonCTSNet*

License Required

Milkyway

dbSetScanChainDefaultPriority

Sets the default priority for scan chain groups. If a scan chain edge constraint references a cell instance that is not explicitly placed in a scan chain group, the Synopsys application creates a scan chain group containing only that cell instance. The application assigns it a priority equal to the default priority (except for the primary input scan cell, which is always at the beginning of the scan chain, and the primary output scan cell, which is always at the end of the scan chain).

Syntax

```
dbSetScanChainDefaultPriority cellId priority "chainName"
```

where the arguments are:

cellId

Database object ID of the cell to which the information applies.

Valid values:

* Cell ID of any open cell in the current library.

* (geGetEditCell)

For information, see Specifying Cells on page 3-57.

priority

Default priority for cell instances that you place in an edge constraint but do not assign to a scan chain group.

Valid values: Any integer from 0 to 160 (0 is the lowest priority and 160 is the highest priority).

chainName

Name of the scan chain for which you are setting a default priority.

Valid values: Name of any scan chain in the open cell.

License Required

Milkyway

5

Scheme Graphic Editor Extensions

This chapter provides descriptions and syntax for Graphic Editor extensions to Scheme. These extensions enhance the languages ability to interact with the graphics editor.

Note:

Unless otherwise specified, the examples are from the Astro product.

geDeselectAllObject

Deselects all previously selected objects.

Syntax

```
geDeSelectAllObject windowNumber
```

where *windowNumber* is the number of the graphics window from which you want to delete selected objects. The argument *windowNumber* can be any positive integer that corresponds to an open window.

See Also

[“geDeselectObject” on page 5-2](#)

[“geSelectObject” on page 5-10](#)

geDeselectObject

Deselects one of the previously selected objects.

Syntax

```
geDeselectObject windowNumber objectID
```

where the arguments are as follows:

windowNumber

Number of the window to which object information applies

Valid values: Any positive integer corresponding to an open window

objectId

Database ID of all object you wish to subtract from the windows selected set

Valid values: Object ID of any object in an open window

See Also

[“geDeselectAllObject” on page 5-2](#)

[“geSelectObject” on page 5-10](#)

geGetEditCell

Returns the cell ID of a cell open in the current graphics window.

Prerequisite

The cell for which you want the cell ID must be opened using the function *geOpenCell*. For more information see *geOpenCell* in the *IC Implementation Command Reference Manual*.

Syntax

`geGetEditCell`

Note:

The function [“dbOpenCell” on page 4-67](#) does not open cells in the graphics window. Therefore, *geGetEditCell* does not recognize those cells.

If you did not use the function *geOpenCell* to open your cells, *geGetEditCell* returns a set of parentheses, `(())`.

See Also

[“geSelectObject” on page 5-10](#) for an example of *geGetEditCell*

[“Specifying Cells” on page 4-49](#)

geGetEnvValue

Returns the current setting of an environment variable for a graphics window.

Syntax

```
geGetEnvValue windowNumber "envName"
```

where the arguments are as follows:

windowNumber

Number of the graphics window that you want information about the environment variables

Valid values: Any positive integer corresponding to an open window or (*hiGetCurrentWindow*)

envName

Name of the environment variable that you want the current setting

Valid values:

- *FilterSize*
- *ViewList*
- *ViewLevels*
- *YMajorGridSpacing*

- *XMajorGridSpacing*
- *YGridSpacing*
- *XGridSpacing*
- *YSnapGrid*
- *XSnapGrid*
- *MaxDragObjects*
- *SelectionAperture*
- *SnapMode*
- *UseLocalName*
- *DisplayMajorGrid*
- *DisplayGrid*
- *SnapToGrid*
- *ShowPathOutline*
- *ShowPathCenter*

geGetSelectedSet

Returns the object IDs for selected objects.

Prerequisites

The cell containing the objects you want to select must be open in a graphics window. In other words, the cell must be opened using *geOpenCell*. For more information see *geOpenCell* in the *IC Implementation Command Reference Manual*. Also, you must select objects in the graphics window.

Syntax

```
geGetSelectedSet windowNumber
```

where *windowNumber* is the number of the window that contains the selected object(s) for which you want ID information. The argument *windowNumber* can be any positive integer corresponding to an open window or (*hiGetCurrentWindow*).

Example

```
geGetSelectedSet 1  
(db:1270011 db:1270013 db:2d1007a db:3f5000b)
```

hiliteNet

Highlights a net. This command is useful for cross probing via the IPC protocol.

Syntax

```
hiliteNet "netName"
```

where *netName* is the name of any net in the current window that you want to highlight.

gelsLayerSelectable

Specifies whether or not a layer can be selected. This function returns the selection option for a layer when *layerNameOrNumber* is specified; otherwise, a list of the selection options for all layers.

Syntax

```
gelsLayerSelectable windowNumber [layerNameOrNumber]
```

where the arguments are as follows:

windowNumber

Number of the graphics window containing the layer for which you want information

Valid values: Any positive integer corresponding to an open window

Note:

If *windowNumber* is 0, *gelsLayerSelectable* returns the default settings that will be applied to new windows.

layerNameOrNumber

Layer for which you want to set selection options

gelsLayerVisible

Specifies whether or not a layer is visible. This function returns the visibility option for a layer when *layerNameOrNumber* is specified; otherwise, a list of the visibility options for all layers.

Syntax

```
geIsLayerVisible windowNumber [layerNameOrNumber]
```

where the arguments are as follows:

windowNumber

Number of the graphics window containing the layer for which you want information

Valid values: Any positive integer corresponding to an open window

Note:

If *windowNumber* is 0, *gelsLayerVisible* returns the default settings that will be applied to new windows.

layerNameOrNumber

Layer for which you want to set visibility options

gelsObjectSelectable

Specifies whether or not an object can be selected. This function returns the selection option for an object when *objectType* is specified; otherwise, a list of the selection options for all objects.

Syntax

```
geIsObjectSelectable windowNumber ["objectType"]
```

where the arguments are as follows:

windowNumber

Number of the graphics window containing the objects for which you want information

Valid values: Any positive integer corresponding to an open window

Note:

If *windowNumber* is 0, *geIsObjectSelectable* returns the default settings that will be applied to new windows.

objectType

Type of object for which you want to set selection options

gelsObjectVisible

Specifies whether or not an object is visible. This function allows greater control of the visibility of objects than the *Options > Window* command. For example, this function allows separate control of rectangles and polygons, while the *Options > Window* command combines the two types as polygon.

This function returns the visibility option for an object when *objectType* is specified; otherwise, a list of the visibility options for all objects.

Syntax

```
geIsObjectVisible windowNumber [ "objectType" ]
```

where the arguments are as follows:

windowNumber

Number of the graphics window containing the objects for which you want information

Valid values: Any positive integer corresponding to an open window

Note:

If *windowNumber* is 0, *getIsObjectVisible* returns the default settings that will be applied to new windows.

objectType

Type of object for which you want to set visibility options

geSelectObject

Adds a windows selected set to a window.

Syntax

`geSelectObject windowNumber objectId`

where the arguments are as follows:

windowNumber

Number of the window from which you select objects

Valid values: Any positive integer corresponding to an open window

objectId

Database ID of the object you wish to select

Valid values: Any object ID of any object in the selected window

Example - Select a cell instance from a selected set.

```
geSelectObject 1  
(dbGetCellInstByName (geGetEditCell) "inst1")  
#t
```

In this example, Astro selects the instance “*inst1*.”

geSetValue

Sets the following environment variable(s) for a graphics window:

- *FilterSize* controls whether or not all objects are visible. Any object smaller than *FilterSize* is not drawn. *FilterSize* is a percent of the screen size.
- *ViewList* controls which cell master is displayed when viewing levels are deeper than 0.
- *ViewLevels* indicates the number of levels of hierarchy displayed in the window during drawing.
- *YMajorGridSpacing*, *XMajorGridSpacing*, *YGridSpacing*, *XGridSpacing*, set grid options.
- *YSnapGrid* and *XSnapGrid* specify the grid to which the cursor snaps.
- *MaxDragObjects* specifies the maximum number of objects that are real-time displayed during certain data modification commands. A value of 0 implies no limit.
- *SnapAperture* controls the maximum distance a coordinate can be moved. This environment variable is currently unused.

- *SelectionAperture*, which is a percent of the screen size, controls the maximum distance to consider when you select objects.
- *SnapMode* is an environment variable used by data entry commands.
- *UseLocalName* controls whether commands use the hierarchical or flat name for nets and cell instances.
- *DisplayMajorGrid* and *DisplayGrid* are boolean values that specify the display options for grids.
- *SnapToGrid* is a boolean value that controls whether the grid is visible.

Note:

Some commands display a grid regardless of the value of *SnapToGrid*.

- *ShowPathOutline* and *ShowPathCenter* are boolean values that control the display of paths.
- *InfixMode* is currently unused.
- *CursorLocation* is currently unused.

Syntax

```
geSetValue windowNumber "envName" value
```

where the arguments are as follows:

windowNumber

Number of the graphics window for which you want information about the environment variables

Valid values: Any positive integer corresponding to an open window

envName

Name of the environment variable for which you want the current setting

Valid values: See the following tables.

value

Value depends on the value of *envName*. See the following tables.

The environment variables in the following table are unique for each window. For example, the same cell may be viewed in two windows at different depths.

envName	value
<i>FilterSize</i>	<i>FilterSize</i> is a percent of the screen size.
<i>ViewLevels</i>	

The environment variables in the following table are unique for each cell. For example, two windows containing the same cell share the value for these environment variables.

envName	value
<i>ViewList</i>	
<i>YMajorGridSpacing</i>	
<i>XMajorGridSpacing</i>	
<i>XGridSpacing</i>	Note: When a library is opened, this value is set to the minimum grid specified in the technology file for the library.
<i>YGridSpacing</i>	
<i>YSnapGrid</i>	
<i>XSnapGrid</i>	

envName	value
<i>UseLocalNames</i>	<p>Valid values:</p> <ul style="list-style-type: none"> • <i>#t</i> requires you to type the flat name for cell instances or nets in all cell instance or net name fields. • <i>#f</i> requires you to type the hierarchical name for cell instances or nets in all cell instance or net name fields.
<i>DisplayMajorGrid</i> <i>DisplayGrid</i>	<p>Valid values:</p> <ul style="list-style-type: none"> • <i>#t</i> displays the grid or major grid. • <i>#f</i> blocks display of the grid or major grid.
<i>SnapToGrid</i>	<p>Valid values:</p> <ul style="list-style-type: none"> • <i>#t</i> blocks display of the grid. • <i>#f</i> displays the grid or major grid. <p>Note: Some commands display the grid regardless of the setting for <i>SnapToGrid</i>.</p>

The environment variables in the following table are global environment variables and apply for all windows. For example, if *ShowPathOutlines* is turned off, all windows will block display of path outlines.

envName	value
<i>MaxDragObjects</i>	<p>Valid values:</p> <ul style="list-style-type: none"> • 0 implies no limit.
<i>SelectionAperture</i>	<i>SelectionAperture</i> is a percent of the screen size.
<i>SnapMode</i>	
<i>ShowPathOutline</i>	<p>Valid values:</p> <ul style="list-style-type: none"> • <i>#t</i> displays the outline of a path. • <i>#f</i> blocks display of the outline of a path.
<i>ShowPathCenter</i>	<p>Valid values:</p> <ul style="list-style-type: none"> • <i>#t</i> displays the center of a path. • <i>#f</i> blocks display of the center of a path.

envName	value
<i>DisplayMajorGrid</i>	Valid values: • <i>#t</i> displays the grid or major grid.
<i>DisplayGrid</i>	• <i>#f</i> blocks display of the grid or major grid.
<i>SnapToGrid</i>	Valid values: • <i>#t</i> blocks display of the grid. • <i>#f</i> displays of the grid or major grid. Note: Some commands display the grid regardless of the setting for <i>SnapToGrid</i> .

If *geSetValue* is successful, it returns *#t*. When *geSetValue* fails, it returns *nil* or *(())*.

geSetLayerSelectable

Specifies whether or not a layer can be selected.

Syntax

geSetLayerSelectable windowNumber layerNameOrNumber select?

where the arguments are as follows:

windowNumber

Number of the graphics window containing the layer for which you want information

Valid values: Any positive integer corresponding to an open window

Note:

If *windowNumber* is 0, *getLayerSelectable* returns the default settings that will be applied to new windows.

layerNameOrNumber

Layer for which you want to set selection options

Valid values:

select?

Indicates whether or not you can select the layer.

Valid values:

- *#t* means you can select the layer.
- *#f* means you cannot select the layer.

If *geSetLayerSelectable* is successful, it returns *#t*. When *geSetLayerSelectable* fails, it returns *nil* or *(())*.

geSetLayerVisible

Specifies whether or not a layer is visible.

Syntax

```
geSetLayerVisible windowNumber layerNameOrNumber visible?
```

where the arguments are as follows:

windowNumber

Number of the graphics window containing the layer for which you want information

Valid values: Any positive integer corresponding to an open window

Note:

If *windowNumber* is 0, *getLayerVisible* returns the default settings that will be applied to new windows.

layerNameOrNumber

Layer for which you want to set visibility options

Valid values:

select?

Indicates whether or not the layer is visible.

Valid values:

- *#t* means the layer is visible.
- *#f* means the layer is not visible.

If *geSetLayerSelectable* is successful, it returns *#t*. When *geSetLayerSelectable* fails, it returns *nil* or *(())*.

geSetObjectSelectable

Specifies whether or not an object can be selected.

Note:

An object can be selected only if both of the following are true:

- The type of the object is selectable.
- The layer of the object is selectable.

Syntax

```
geSetObjectSelectable windowNumber "ObjectType" select?
```

where the arguments are as follows:

windowNumber

Number of the graphics window containing the objects for which you want information

Valid values: Any positive integer corresponding to an open window

Note:

If *windowNumber* is 0, *getIsObjectSelectable* returns the default settings that will be applied to new windows.

objectType

Type of object for which you want to set selection options

Valid values:

select?

Indicates whether or not you can select the object type.

Valid values:

- *#t* means you can select the object type.
- *#f* means you cannot select the object type.

If *geSetObjectSelectable* is successful, it returns *#t*. When *geSetObjectSelectable* fails, it returns *nil* or *(())*.

geSetObjectVisible

Specifies whether or not an object is visible.

Note:

An object is visible only if both of the following are true:

- The type of the object is visible.
- The layer of the object is visible.

Syntax

```
geSetObjectVisible windowNumber "objectType" visible?
```

where the arguments are as follows:

windowNumber

Number of the graphics window containing the objects for which you want information

Valid values: Any positive integer corresponding to an open window

Note:

If *windowNumber* is 0, *getIsObjectVisible* returns the default settings that will be applied to new windows.

objectType

Type of object for which you want to set visibility options

Valid values:

select?

Indicates whether or not the object type is visible.

Valid values:

- `#t` means the object type is visible.
- `#f` means the object type is not visible.

If `geSetObjectVisible` is successful, it returns `#t`. When `geSetObjectVisible` fails, it returns `nil` or `(())`.

6

Scheme Floorplanner Extensions

This chapter provides descriptions and syntax for Floorplanner extensions to Scheme.

Note:

Unless otherwise specified, the examples are from the Astro product.

fpGetFPOption

Gets the current value of a floorplanning property.

To set the default options for place and route operations, the *Options > Place & Route* parameters, use “[fpSetFPOption](#)” on page 6-4.

Syntax

```
fpGetFPOption windowNumber "optionName"
```

where the arguments are as follows:

windowNumber

Number of the window in which the cell is located

Valid values: Any integer corresponding to an open window

optionName

Name of the property for which you want the current value

Valid values:

- *Drag Flylines*
- *Snap to Row*
- *Max Pins/Net*
- *Row Alignment*
- *Use Y First*
- *Balance Slice*
- *Maintain Topography*
- *Delete Filler*

- *Route Mode*
- *Compact Channel*
- *Optimize After*
- *Reset Origin*
- *No M2 Strap overlap*
- *Default Core Power Net*
- *Default Core Ground Net*
- *Default Core Clock Nets*
- *Freeze Footprint*
- *Freeze Silicon*

Note:

If you want the current values for all of the floorplanning options, specify “*ALL*” as the *optionName*.

Example - List current values for all floorplanning options.

```
fpGetFPOption 1 "ALL"
(("Default Core Clock Nets" ()))
("Default Core Ground Net" "VSS")
("Default Core Power Net" "VDD")
("Freeze Footprint" #f) ("Freeze Silicon" #f)
("No M2 Strap overlap" #f) ("Optimize After" #t)
("Compact Channel" #t) ("Delete Filler" "Routing")
("Route Mode" "3")
("Reset Origin" #t)
("Maintain Topology" #t) ("Balance Slice" #t)
("Use Y First" #f)
("Max Pins/Net" 0) ("Row Alignment" "Ground")
("Snap To Row" #f) ("Drag Flylines" #f))
```

See Also

[“fpSetFPOption” on page 6-4](#)

fpSetFPOption

Sets the default place and route options via scheme.

Note:

The function fpSetFPOption does not preset any of the floorplanning values; it sets the Options > Place & Route parameters for the current Synopsys session.

To display the values for current option settings, use
[“fpGetFPOption” on page 6-2](#).

Syntax

```
fpSetFPOption windowNumber "optionName" value
```

where the arguments are as follows:

windowNumber

Number of the window in which the object is located

Valid values: Any integer corresponding to an open window

Note:

Zero (0) corresponds to a new window.

optionName

The options available depend on the *windowNumber* selected.

See the following tables.

value

Value depends on the value of *optionName*. See the following tables.

The following combinations apply for all *windowNumber* values.

optionName	value
<i>Drag Flylines</i>	Indicates whether or not flylines are visible. Valid values: <ul style="list-style-type: none">• <i>#t</i> displays the flylines.• <i>#f</i> hides the flylines.
<i>Snap to Row</i>	Indicates whether or not the Synopsys application automatically snaps and adjusts rows when the cell is moved. Valid values: <ul style="list-style-type: none">• <i>#t</i> automatically snaps and adjusts the row when the cell is moved.• <i>#f</i> does not automatically snap and adjust the row when the cell is moved.
<i>Max Pins/Net</i>	Maximum number of pins you want the displayed nets to have when <i>Drag Flylines</i> is turned on. If a net has more pins than the value specified, the net is not displayed.
<i>Row Alignment</i>	Sets the default alignment options for floorplanning. Valid values: <ul style="list-style-type: none">• <i>LowerLeft</i> aligns the bottom of each cell with the bottom of the cell row.• <i>UpperLeft</i> aligns the top of each cell with the top of the cell row.• <i>Origin</i> aligns the origin of each cell with the bottom of the cell row.• <i>Center</i> aligns the center of each cell with the center of the cell row.• <i>Power</i> aligns the top of the power pin in each cell with the top of the cell row.• <i>Ground</i> aligns the bottom of the ground pin in each cell with the bottom of the cell row.• <i>CenterPG</i> aligns the point halfway between the bottom of the power pin and the top of the ground pin in each cell with the center of the cell row.

optionName	value
<i>Use Y First</i>	<p>Indicates whether the Synopsys application slices vertically or horizontally first.</p> <p>Valid values:</p> <ul style="list-style-type: none"> • <i>#t</i> means that the Synopsys application slices vertically first. • <i>#f</i> means that the Synopsys application slices horizontally first. <p>This is the inverse of <i>Use X for first cut</i> on the Place and Route Options Form.</p>
<i>Balance Slice</i>	<p>Indicates whether or not the Synopsys application balances the slicing structure.</p> <p>Valid values:</p> <ul style="list-style-type: none"> • <i>#t</i> balances the slicing structure. • <i>#f</i> does not balance the slicing structure.
<i>Maintain Topology</i>	<p>Indicates whether or not the Synopsys application maintains the current slicing structure.</p> <p>Valid values:</p> <ul style="list-style-type: none"> • <i>#t</i> maintains the current slicing structure. • <i>#f</i> does not maintain the current slicing structure.
<i>Delete Filler</i>	<p>Indicates whether the Synopsys application deletes the filler cells prior to placement or routing.</p> <p>Valid values:</p> <ul style="list-style-type: none"> • <i>Placement</i> deletes filler cells prior to placement. • <i>Routing</i> deletes filler cells prior to routing.
<i>Route Mode</i>	<p>Valid values:</p> <ul style="list-style-type: none"> • 2 indicates two layer metal routing. • 3 indicates three layer metal routing.
<i>Compact Channel</i>	<p>Indicates whether or not the Synopsys application compacts the channels.</p> <p>Valid values:</p> <ul style="list-style-type: none"> • <i>#t</i> means that the channel is compacted. • <i>#f</i> means that the channel is not compacted.

optionName	value
<i>Optimize After</i>	<p>Indicates whether or not the Synopsys application should perform post-route optimization (which reduces wire length and eliminates unnecessary vias).</p> <p>Valid values:</p> <ul style="list-style-type: none"> • <i>#t</i> means that the Synopsys application optimizes the cell after detail routing. • <i>#f</i> means that the Synopsys application does not optimize the cell after detail routing.
<i>Reset Origin</i>	<p>Indicates whether or not the Synopsys application resets the origin to the lower left corner (0,0) after detail-routing of a single-region design or inter-region detail-routing of a multiple-region design.</p> <p>Valid values:</p> <ul style="list-style-type: none"> • <i>#t</i> resets the origin after routing. • <i>#f</i> does not reset the origin after routing.
<i>No M2 Strap overlap</i>	<p>Indicates whether the Synopsys application may place cells beneath metal2 straps</p> <p>Valid values:</p> <ul style="list-style-type: none"> • <i>#t</i> means that the Synopsys application refrains from placing any cells beneath metal2 straps. • <i>#f</i> means that metal2 straps and cells can overlap.
<i>Freeze Footprint</i>	<p>Indicates whether the Synopsys application may modify pin and pad locations of the top-level cell.</p> <p>Valid values:</p> <ul style="list-style-type: none"> • <i>#t</i> means that the Synopsys application attempts to maintain pin and pad locations of the top-level cell. • <i>#f</i> means that pin and pad locations are able to change.
<i>Freeze Silicon</i>	<p>Indicates whether the Synopsys application may modify pin and pad locations of the top-level cell and placement of standard and macro cells.</p> <p>Valid values:</p> <ul style="list-style-type: none"> • <i>#t</i> means that the Synopsys application attempts to maintain pin and pad locations of the top-level cell and placement of standard and macro cells. • <i>#f</i> means that cell placement is able to change.

The following combinations only apply for *windowNumber* !=0.

optionName	value
<i>Default Core Power Net</i>	Name of the power net for which you want to generate straps during automatic floorplanning operations Valid syntax: <ul style="list-style-type: none">• <i>netName</i>• "
<i>Default Core Ground Net</i>	Name of the ground net for which you want to generate straps during automatic floorplanning operations Valid syntax: <ul style="list-style-type: none">• <i>netName</i>• "
<i>Default Core Clock Nets</i>	Name(s) of the clock net(s) for which you ant to generate straps during automatic floorplanning operations Valid syntax: <ul style="list-style-type: none">• '({ "netName" } ...)• <i>netName</i>• '()

Note:

When *windowNumber* is set to 0, only the values of Drag Flylines and MaxPins/Net always copy to the new window. If an existing place and route cell is open, the values for that cell become the default for any new cell.

Example

```
fpSetFPOption 0 "Route Mode" "2"  
#t  
fpSetFPOption 0 "Delete Filler" "Placement"  
#t  
fpSetFPOption 0 "Optimize After" #t  
#t  
fpSetFPOption 0 "Reset Origin" #t  
#t  
fpSetFPOption 0 "Row Alignment" "CenterPG"  
#t
```

See Also

[“fpGetFPOption” on page 6-2](#)

7

Synopsys Scheme Examples

This chapter presents examples that show the use of Synopsys Scheme extensions for IC design.

Beginning Scheme Examples

The following examples provide an introduction to using Scheme for the Milkyway Database Environment. Comments in the code help to explain the flow of the code.

Example 1 - Find Open Library Name

This example opens the library and gets the name of the currently open library.

```
; dbGetCurrentLibId returns the library ID which is opened  
; From this library ID, dbFetchObjectField fetches the  
"libraryName"  
; and returns the name of the library  
  
define _libName (dbFetchObjectField (dbGetCurrentLibId) '()  
"libraryName")
```

Example 2 - Cell Nets

This example opens the library and cell. It then prints all the nets in the cell.

```
; Open the library and cell. This example prints all the nets in the cell.  
  
(define (getAllNetsInCell)  
  (define cellId (geGetEditCell))  
  (begin  
    ; Cycle through the cellId and print all the nets  
    (db-foreach cellId '() "Net" NameId  
      (begin  
        (let (  
          (netName (dbFetchObjectField cellId NameId "name")) )  
        )  
          (display netName) (newline)  
        )
```

```
        #t
    )
)
)
```

Example 3 - Objects of Specific Type

This example displays all objects of a specific type. In this example, it is assumed that you have a cell open in the graphics window. Therefore, you open your current cell by using geOpenCell).

```
define displayObjects (lambda (type)
(define xxx (geGetEditCell))
(db-foreach xxx '() type z
(begin
(print
(dbFetchObject xxx z)
)
#t
)
)
)
```

Example 4 - Instances of Open Cell

This example open the library and cell. It finds all cell instances for the cell which is opened.

```
; Open the Library and Cell. This example finds all cell
instances
; for the cell that is opened.

(define (get_all_cells)
  (let* ( (cellnames '())
         (begin
           ;Cycle through all cell instance and print names of
the cell instances
           (db-foreach (geGetEditCell) '() "CellInst" CELL
           (begin
             (set! cellnames
               ; Get the names of the cell instances in the opened cell
               (cons (dbFetchObjectField (geGetEditCell) CELL
"name") CELLNAMES)
             )
             #t
           )
           (display cellnames)
         )
       )
     )
)
```

Example 5 - Cell Names for a View

This example opens the library and returns a list of cell names for the particular view in your library. The input of the program are the database ID of the library and the name of the view for which you want to print the cell names.

```
; Open the library. This example returns a list of cell names
; for the
; particular view in your library. The input of the program
; are database
; ID of the library and the name of the view for which you
; want to print
; the cell names.

define getAllCellsOfViewInLib (lambda (libId viewName)
  (let ((cellNameList '()))
    ; Cycle through the library
    (db-foreach libId '() "CellName" cellNameRec
      (let*
        (
        ; Get the names of the cells in the library
        (name (dbFetchObjectField libId cellNameRec "cellName"))
        ; Get the view name of the cells in the library
        (view (dbFetchObjectField libId cellNameRec "viewName")))
        (fullCellName (string-append name view)))
      )
      (if (equal? view viewName)
        (set! cellNameList (cons fullCellName cellNameList))
      )
      #t
    )
    )
    cellNameList
  )
)
```

Example 6 - Cell Instances Assigned to a Row

Finds all cell instances assigned to a row that are not specified as standard cells.

```
define FindInst
  (lambda (cellId)
    (db-foreach
      cellId
      '()
      "cellregion"
      regionId
      (begin
        (display (cons "Processing region" regionId))
        (db-foreach cellId regionId "CellRow" rowId
          (begin
            (db-foreach
              cellId
              rowId
              "cellInst"
              instId
              (begin
                (begin
                  (define cflag
                    (dbFetchObjectField cellId instId "cellFlag"))
                  )
                  (if
                    (equal? 0 (bit-and cflag 0x4000))
                    (display
                      (dbFetchObject cellId instId)
                      )
                    #t
                  )
                  #t
                )
                #t
              )
            )
            #t
          )))
        )
      )
    )
  )
```

Intermediate Scheme Examples

The following examples build on the beginning Scheme examples. Comments in the code help to explain the steps in the script.

Example 5 - Bounding Box of Cell Instances

This example opens the library and cell. It then takes the opened cell and cycles through all instances printing the bounding box to a file specified.

```
; Open the library and cell. This function takes the opened cell, and cycles
; through all instances printing the bounding box to a file specified.
(define dumpCellInstBB (lambda (fileName)
    ;Open Output file
    (define port (open-output-file fileName))
    ;Get Cell Id
    (define cellId (geGetEditCell))
    ;cycle through all instance and print name and bounding box
    (db-foreach cellId '() "CellInst" instanceId
        (begin
            ;get info from cell instance
            (define name (dbFetchObjectField cellId instanceId "cellInstanceName"))
            (define cellBoundary (dbFetchObjectField cellId instanceId "cellboundary"))

            ;get x an y of boundary
            (define x1 (car (car (car (cdr cellBoundary)))))
            (define x2 (car (car (cdr (car (cdr cellBoundary))))))
            (define x3 (car (car (cdr (cdr (car (cdr cellBoundary)))))))
            (define x4 (car (car (cdr (cdr (car (cdr cellBoundary)))))))

            (define y1 (car (cdr (car (car (cdr cellBoundary))))))
            (define y2 (car (cdr (car (cdr (car (cdr cellBoundary)))))))
            (define y3 (car (cdr (car (cdr (cdr (car (cdr cellBoundary))))))))
            (define y4 (car (cdr (car (cdr (cdr (car (cdr cellBoundary)))))))

            ;find min and max x and y
            (define minx x1)
            (if (> minx x2)
                (set! minx x2)
            )
```

```

(if (> minx x3)
(set! minx x3)
)
(if (> minx x4)
(set! minx x4)
)

(define miny y1)
(if (> miny y2)
(set! miny y2)
)
(if (> miny y3)
(set! miny y3)
)
(if (> miny y4)
(set! miny y4)
)

(define maxx x1)
(if (< maxx x2)
(set! maxx x2)
)
(if (< maxx x3)
(set! maxx x3)
)
(if (< maxx x4)
(set! maxx x4)
)

(define maxy y1)
(if (< maxy y2)
(set! maxy y2)
)
(if (< maxy y3)
(set! maxy y3)
)
(if (< maxy y4)
(set! maxy y4)
)

;construct points
(define pt1 (cons minx (list miny)))
(define pt2 (cons maxx (list maxy)))
(define bBox (cons pt1 (list pt2)))

(display "Instance Name: " port)
(display name port)
(display " BBox: " port)

```

```

(display bBox port)
(newline port)
#t
)
)
;close output port
(close-output-port port)
;exit
(display "Dump Cell Instance Bounding Box Program finished.")
(newline)
#t))

```

Example 6 - Fanout Search

```

; This function takes a Fanout number from the user and scans the
; design for nets that have a Fanout >= to the user's number.

(define GenFanReport (lambda (cellId Fanout)
  (define Fcount 0)
  (define PName (getProgramName))
  (let*
    (
      (db-null (CONSTANT "db-null"))
    )
    (display "Checking Nets with Fanout >= ")
    (display Fanout)
    (newline)
    (db-foreach cellId '() "net" parentNetId
      (begin
        (define Netname (dbFetchObjectField cellId parentNetId "name"))
        ; Scan parent net
        (db-foreach cellId parentNetId "netMember" memberId
          (begin
            ; Do it for Each member of the parent net
            (SumUpFanout cellId memberId )
            #t
          )
        )
        (if (>= Fcount Fanout)
          (begin
            (display Netname)
            (display " ")
            (display Fcount)
            (newline)
            #t
          )
        )
      )
    )
  )

```

```

)
'()
)
(set! Fcount 0 )
#t
);close begin
);close db-for

);closes let*
#t))
)

(define SumUpFanout (lambda (cellId netMemberId )
(let
((recordType (dbFetchObjectField cellId netMemberId "recordType")))
(if (string=? recordType "PortInst")
(set! Fcount (+ Fcount 1))
'()
)
)
#t)))

```

Example 7 - Pins and Ports Checker

```

; This function is a simple checker that looks for differences between pins
; and ports in a FRAM view. The program will optionally create a
; file of cellnames, portlist and pinlist.
;

;
;           dbFindPortPinMismatchInLib "reflib" #f
;           dbFindPortPinMismatchInLib "newlib" #t
;

;
;       where the arguments are:
;           1) The Library to be checked. ( a string   )
;           2) #f Do not create file of port/pin list
;           3) #t Create file "Libname_port-pin_list" that
;                  contains a listing of all cells and their
;                  corresponding port and pin list.
;

;
;

(define dbFindPortPinMismatchInLib (lambda (libname fileflag)
(let
(
(cellId 0)

```

```

(libId 0)
(err 0)
(masterportlist '())
(masterpinlist '())
(viewName "")
(nameLength "")
(inLib #f)
(fptra 0)
)
(newline)
(set! libId (dbOpenLibrary libname "r"))
(if (dbId? libId)
(begin
(display "Scanning Library ")
(display libname)
(display " for Port/Pin MisMatches..... ")
(newline)
(if (eq? fileflag #t)
(begin
(set! fptra (open-output-file (string-append libname "_port-
pin_list")))
(display ";" fptra) (newline fptra)
(display ";" Port/Pin File for Library --> " fptra)
(display libname fptra)
(newline fptra)
(display ";" fptra) (newline fptra)
(display ";Format Cellname ---> Port List ---> Pin List " fptra)
(newline fptra)
(newline fptra)
(display "Writing Library Port/Pin Summary to file --> ")
(display (string-append libname "_port-pin_list"))
(newline)
)
)
;
; checking the difference between the ports and pins
; for all the cells in FRAM view
(for-each
(lambda (cellName)
(set! nameLength (string-length cellName))
(set! viewName (substring cellName (- nameLength 5) nameLength))
(if (string=? viewName ".FRAM")
(begin
(set! cellId (dbOpenCell cellName "r"))
(set! masterportlist (findAllPortsInCell cellId))
(set! masterpinlist (findAllPinsInCell cellId))
(if (eq? fileflag #t)
(begin
(display cellName fptra)

```

```

        (display " ---> " fptr)
        (display masterportlist fptr)
        (display " ---> " fptr)
        (display masterpinlist fptr)
        (newline fptr)
    )
)
(if (not (= (length masterportlist) (length masterpinlist)))
    (begin
        (newline)
        (display "!!! Mismatch Found in Library !!!")
        (newline)
        (display "Cell ")
        (display cellName)
        (display " with Ports -> ")
        (display masterportlist)
        (display " with Pins -> ")
        (display masterpinlist)
        (newline)
    )
)
(for-each
    (lambda (Portname)
        (set! inLib (member Portname masterpinlist))
        (if (eqv? inLib #f)
            (begin
                (display "Cell ")
                (display cellName)
                (display " is missing Port ")
                (display Portname)
                (display " in Cell's Pinlist ")
                (newline)
                (set! err 1)
            )
        )
    )
)
masterportlist
)
(for-each
    (lambda (Pinname)
        (set! inLib (member Pinname masterportlist))
        (if (eqv? inLib #f)
            (begin
                (display "Cell ")
                (display cellName)
                (display " is missing Pin ")
                (display Pinname)
                (display " in Cell's Portlist ")
                (newline)
            )
        )
    )
)

```

```

                (set! err 1)
            )
        )
    )masterpinlist
)
(if (= err 1)
(begin
    (display "!!! End of Cell Error !!!")
    (newline)
    (newline)
    (set! err 0)
)
)
(dbCloseCell cellId)
);begin
)
) (getAllCellNamesInLib libId)
)
(dbCloseLib libId)
(if (eq? fileflag #t)
    (close-output-port fptr)
)
(newline)
(display "Done ... ")
(newline)
);begin
(begin
    (display "Could not Open Library -> ")
    (display libname)
    (newline)
    (display "Program Abort")
    (newline)
)
);if (dbId? libId)
)
#t))
; getAllCellNamesInLib reports all the cells for all the views
; in the library which is opened

(define getAllCellNamesInLib (lambda (libId)
(let ((cellNameList '()))
; Cycle through the libId and report all the cell names in all the views
(db-foreach libId '() "CellName" cellNameRec
(let*
(
(name (dbFetchObjectField libId cellNameRec "cellName"))
(view (dbFetchObjectField libId cellNameRec "viewName")))

```

```

        (fullCellName (string-append name view))
    )
    (set! cellNameList (cons fullCellName cellNameList))
    #t
)
)
cellNameList
)
))

; findAllPortsInCell reports all the ports in the cell which is opened
(define findAllPortsInCell (lambda (cellId)
  (let ((portlist '()))
    ; Cycle through the cellId and report all ports
    (db-foreach cellId '() "Port" thisPortId
      (let
        (
          (portName (dbFetchObjectField cellId thisPortId "portName"))
        )
        (set! portlist (cons portName portlist))
        #t
      )
    )
    portlist
  )
))

; findAllPinsInCell reports all the pins in the cell which is opened
(define findAllPinsInCell (lambda (cellId)
  (let ((pinlist '()))
    ; Cycle through the cellId and report all the pins
    (db-foreach cellId '() "Pin" thisPinId
      (let
        (
          (pinName (dbFetchObjectField cellId thisPinId "Name"))
        )
        (set! pinlist (cons pinName pinlist))
        #t
      )
    )
    pinlist
  )
)))

```

8

Using the Milkyway Scheme Debugger

This chapter describes the Milkyway Scheme Debugger, which provides a debugging environment and a graphical interface in which you can debug Scheme programs. It includes the following sections:

- [Introduction](#)
- [Getting Started](#)
- [Debug Commands](#)

Introduction

The Milkyway Scheme Debugger is a dynamic extension of the Synopsys Scheme language. It provides a debugging environment and a graphical interface in which you can debug Scheme programs at the source level.

The debugger also provides a new way of replaying command files. It allows you to interactively replay a command file in a controlled manner and to experiment with different command options along the way. This is a useful complement to executing your Synopsys tool in batch mode with the `-replay` option.

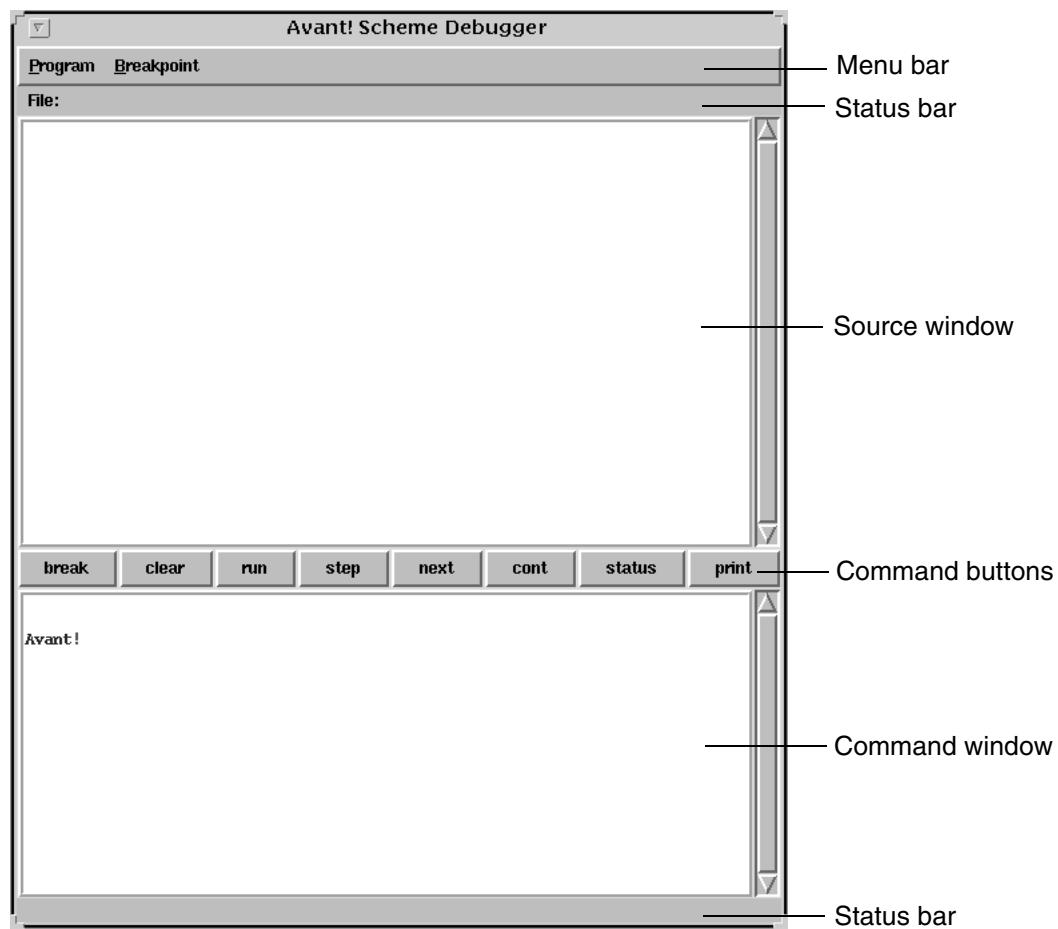
The debugger is not a stand-alone program but is available in any Synopsys tool that has an interactive Scheme interface. You invoke it from within the Scheme environment of your Synopsys tool. The debugger has its own graphical user interface (GUI), which consists of the following components:

- Source window: The source window displays the current source code and shows the breakpoints and execution stop points of the program, which are automatically updated as the debugging proceeds. Using your mouse, you can also set and clear breakpoints and select text (for printing) in the source window.
- Command window: The command window enables you to interact with the debugger. You have full control of the debugger when typing commands in the command window. The debugger displays its response in the command window and updates the source window when needed.

- Command buttons: A row of command buttons is located between the source and command windows. These buttons provide shortcuts to some of the commonly used debugging commands and allow you to control the debugging actions by using your mouse.
- Menu bar: The two menus at the top of the interface, Program and Breakpoint, provide additional mouse-driven control over some of the debugger functions.
- Status Bars: There are two status bars. The one at the top of the source window, labeled File, shows the current file name. The one at the bottom of the command window displays a simple instruction about the mouse function at the current position of the pointer.

[Figure 8-1](#) shows the Scheme Debugger interface.

Figure 8-1 Scheme Debugger GUI



When the debugger is running, the main Scheme application window from which you started the debugger is still active and all the Scheme commands are accessible from there. The debugger starts a new Scheme debugging environment without turning off the original Scheme environment. These two environments share the same internal Scheme evaluator as well as the external data on your hard drive.

Note:

The Scheme evaluator runs slightly slower in debug mode.

Getting Started

This section contains the following topics:

- [Starting the Debugger](#)
- [Interacting With the Debugger](#)
- [Setting and Clearing Breakpoints](#)
- [Debugging a Program](#)
- [Quitting the Debugger](#)

Starting the Debugger

You invoke the debugger from within the Scheme application window by entering the following Scheme command:

```
(debug 'filename)
```

The *filename* argument is optional. If it is provided, the source file will be displayed in the source window and will be set as the default debugging program in the debug session. It will remain the default until you explicitly change it by running the load or run command later in the debugging process.

Interacting With the Debugger

You can interact with the debugger in the command window. All the debug information is displayed in that window. However, any output from the Scheme program being debugged is displayed where it

normally is when run outside the debugger. This is typically the main application's Scheme command window, from which you started the debug session.

Setting and Clearing Breakpoints

You can set breakpoints by using the `break [filename] line` command (see “[break \[filename\] line](#)” on page 12) or the break button. Breakpoints are set in the currently displayed source file and marked in yellow. If a program is made up of more than one source file, you can use the file command to display the file and then set or clear breakpoints in it as usual.

You clear breakpoints by using the `clear [filename] line` and `clearall` commands and the clear button.

- The `clear [filename] line` command clears a single breakpoint in the current displayed file.
- The `clearall` command clears all the breakpoints in all the files previously used.

It is recommended that you use `clearall` before starting to debug a new program.

Debugging a Program

To start a program in the debugger, use the `run [filename]` command where `filename` contains the program to debug. Providing the `filename` argument is optional. If you provide it, the program is executed; `filename` is also set as the default main program, so that the next time the `run` command is issued without an argument, `filename` will be the default.

The execution stops if an error occurs or if a break point is encountered. When a program stops for either reason, the stop position is marked in orange in the source window. If the program stopped because of an error, an error message is displayed in the command window.

If the program stopped at a breakpoint, you can restart it by clicking the step, next, or cont command button. Which button you use depends on how you want the program to be stopped again.

- Use step to stop the program at the next Scheme expression.
- Use next to stop the program at the next line of code.
- Use cont to stop the program at the next breakpoint, if any is encountered.

At any stop, you can examine the status of the program, set or clear breakpoints, or check the values of variables (by using the `print exp` command). You can modify variable values (by using the `assign var exp` command) to change the execution direction of the program. For more information, see “[assign var exp](#)” on page 14.

Quitting the Debugger

To quit the debugger, use the `debug exit` or `quit` command. The debug process terminates, the debugger user interface closes, and you are returned to the top Scheme command window.

You can also quit the debugger by using menu commands: Choose Program>Exit.

Note:

Quitting the debugger does not automatically close any libraries and cells that were opened during the debug session.

Debug Commands

Debug commands fall into four functionality groups, as described in the following sections:

- [File Loading Commands](#)
- [Execution Commands](#)
- [Breakpoint Commands](#)
- [Other Commands](#)

File Loading Commands

There are two file loading commands:

- `file filename`
- `load [filename]`

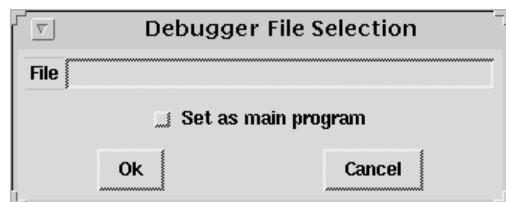
file filename

The `file filename` command loads the file you specify into the source window, where you can examine it and set breakpoints. If the file has previously been accessed and breakpoints have been set, the old breakpoints are marked. Note that if the file has been modified, the old breakpoints might be set on the wrong lines.

GUI equivalent:

- Button: None.
- Menu: Program>File. Opens the Debugger File Selection dialog box, in which you can type a file name (see [Figure 8-2](#)). Be sure the “Set as main program” button is not selected.

Figure 8-2 Debugger File Selection Dialog Box



load [filename]

The `load` command is similar to `file`, except that the debugger resets the default execution program to *filename*. With this command the *filename* argument is optional. If you do not specify an argument, the current default program is loaded into the source window.

GUI equivalent:

- Button: None.
- Menu: Program>File. Opens the Debugger File Selection dialog box, in which you can type a file name. Be sure the “Set as main program” button is not selected.

Execution Commands

There are four execution commands:

- `run filename`
- `step [n]`
- `next`
- `cont`

run [filename]

You use the `run [filename]` command to begin executing a program.

If you provide a *filename* argument, the debugger loads the file into the source window, sets it as the default program, and executes the program.

If you do not specify a file name, the debugger loads and executes the current default program, which was set by the latest `run [filename]` or `load [filename]` debugger commands or by the `(debug 'filename)` Scheme command.

GUI equivalent:

- Button: The run button executes the current default program.
- Menu: None.

step [n]

The `step` command executes the next *n* Scheme expressions, starting from the current expression. Any breakpoints set in the next *n* expressions are ignored. If you do not set the *n* argument, the debugger uses the default, which is 1. If the ending expression in the evaluation path is defined in a different file, that file is loaded in the source window first.

GUI equivalent:

- Button: The step button is equivalent to `step 1`.
- Menu: None.

next

The `next` command executes all the expressions in the current source line and stops at the next line in the same file. However, if one of the expressions leads to the execution of commands defined in another file, that file is loaded into the source window and the execution stops at the appropriate line within it.

GUI equivalent:

- Button: The next button is equivalent to `next`.
- Menu: None.

cont

The `cont` command continues execution from the current stop point until the next breakpoint.

GUI equivalent:

- Button: The cont button is equivalent to `cont`.
- Menu: None.

Breakpoint Commands

There are four breakpoint commands:

- `break [filename] line`
- `clear [filename] line`
- `clearall`
- `status`

break [filename] line

The `break [filename] line` command sets a breakpoint at a line in the file you specify. If there is only one argument, it is taken to be `line`, and the breakpoint is set in the currently displayed file.

GUI equivalent:

- Button: Select a line in the source window then click the break button.
- Menu: Select a line in the source window, then choose Breakpoint >Break.

clear [filename] line

The `clear [filename] line` command clears a breakpoint at a line in the file you specify. If there is only one argument, it is taken to be `line`, and the debugger clears the breakpoint at `line` in the currently displayed file.

GUI equivalent:

- Button: Select a line in the source window then click the clear button.
- Menu: Select a line in the source window, then choose Breakpoint>Clear.

clearall

The `clearall` command clears all breakpoints in previously accessed files.

GUI equivalent:

- Button: None.
- Menu: Breakpoint>Clear all.

status

The `status` command prints the file name of the default program, the running status, and the breakpoints set in all files.

GUI equivalent:

- Button: The status button is equivalent to `status`.
- Menu: Breakpoint>`status`.

Other Commands

The following additional Scheme debugger commands are available:

- `assign var exp`
- `print exp`
- `search string`
- `bsearch string`
- `cd dir`
- `pwd`
- `quit`
- `exit`

assign var exp

The `assign var exp` command assigns the value of the Scheme `exp` expression to the `var` variable. If `var` is an existing variable in the current environment, its value is updated. Otherwise a new global variable is created.

GUI equivalent: None.

print exp

The `print exp` command executes the Scheme `exp` expression and prints out the result in the command window. You can use this command simply to examine the values of variables at a stop or, more generally, to run a complicated Scheme expression.

GUI equivalent:

- Button: Select a variable or expression in the source window, then click the print button.
- Menu: None.

search string

The `search string` command searches forward for the specified string in the currently displayed file, where `string` can be a regular expression. If found, the text string is marked in the source file.

Normally the search starts from the beginning of the file—unless the last `search` command searched and found the same string. In that case, the current search starts right after its last location.

GUI equivalent:

- Button: None.

- Menu: Program>Search in File. Opens the Debugger Search dialog box (see [Figure 8-3](#)), in which you can type a string, select Down for forward searching, and then click OK to begin the search.

Figure 8-3 Debugger Search Dialog Box



bsearch string

The `bsearch string` command is the same as the `search string` command, except that `bsearch` searches backward for `string`.

GUI equivalent:

- Button: None.
- Menu: Program>Search in File. Opens the Debugger Search dialog box (see [Figure 8-3 on page 8-15](#)), in which you can type a string, select Up for backward searching, and then click OK to begin the search.

cd dir

The `cd dir` command changes the current working directory to `dir`.

GUI equivalent: None.

pwd

The `pwd` command shows the current working directory.

GUI equivalent: None.

quit

The `quit` command terminates the debug session, closes the debugger interface, and returns you to the top Scheme command window.

GUI equivalent:

- Button: None.
- Menu: Program>Exit.

exit

The `exit` command is the same as `quit`.

9

Customizing Milkyway

Milkyway provides customizations to address your specific design flow. This chapter covers the following topics:

- User-Defined Menus and Dialog Boxes
- Attached File Management

Creating Milkyway Menus and Dialog Boxes

The Tcl scripting language and the Tk toolkit (Tcl/Tk) provide a programming environment for building graphical user interfaces that can run on multiple platforms. Using Scheme and Tcl/Tk, you can quickly develop menus and dialog boxes in the Milkyway GUI to suit your specific design requirements.

This section discusses programming the Tcl/Tk Scheme interface to develop custom Milkyway menus and dialog boxes. It includes the following:

- Creating Menus by Using Tcl/Tk
- Tcl/Tk Scheme Interface
- Creating Menus in Milkyway

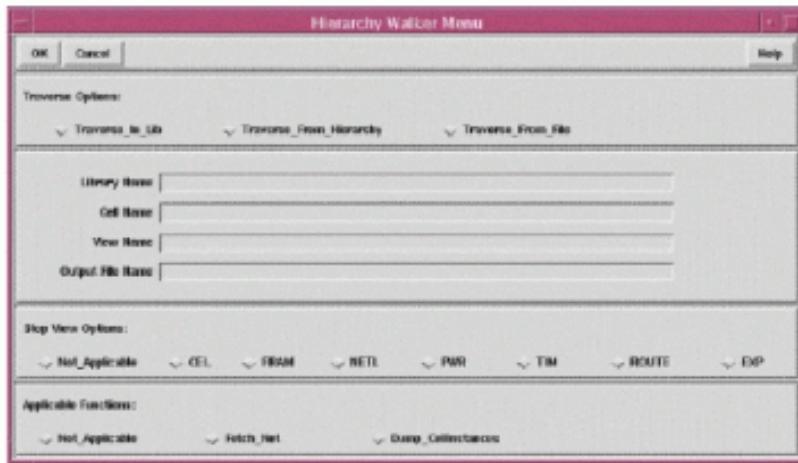
Creating Menus by Using Tcl/Tk

The Tcl scripting language and the Tk toolkit provide a programming environment for building graphical user interfaces that can run on multiple platforms. With a short amount of coding, you can integrate Tcl/Tk with Scheme to develop user interfaces.

Tcl/Tk provides convenient built-in applications to ease GUI development. The purpose of this section is to give a quick introduction to creating menus and dialog boxes with Tcl/Tk. For more details on the Tcl/Tk language, see one of the many published references.

The following sections use the Tcl program myMenu.tcl for all examples and references. A complete script of myMenu.tcl is provided at the end of this document (see [Example 9-8 on page 9-15](#)).

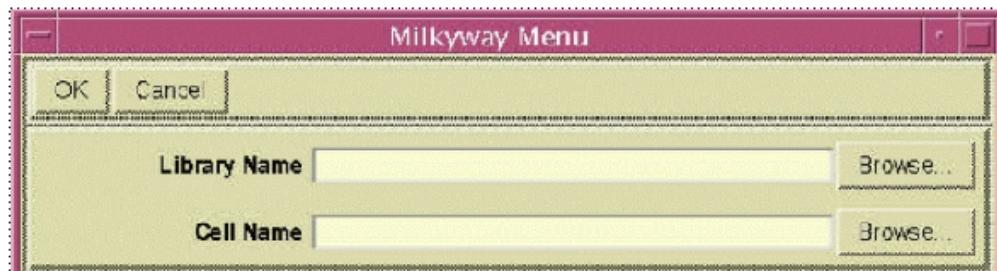
The Tcl/Tk program, myMenu.tcl creates a dialog box that looks like the following:



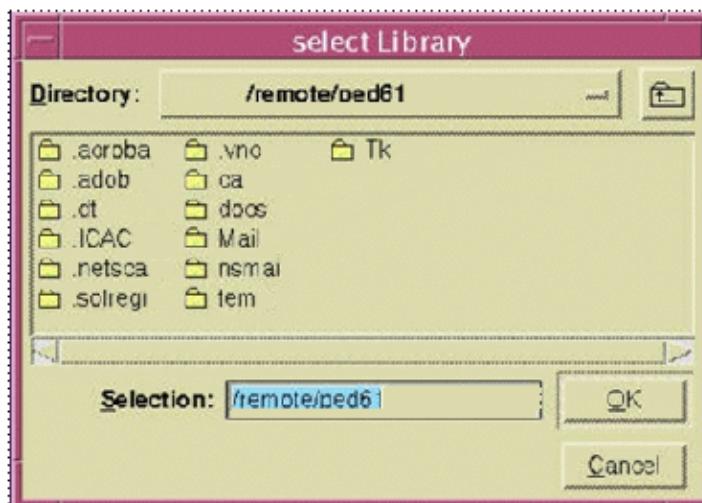
One advantage of using Tcl/Tk is its convenient built-in applications, which facilitate GUI development. For example, Tcl/Tk provides a browser application with command that allow users to make selections from a dialog box.

Specifically, by using two Tcl/Tk commands you can select a directory name (`tk_chooseDirectory`) and a cell name (`tk_getOpenFile`). An example of a browser application follows, together with the Tcl/Tk code used to create it.

The Tcl/Tk code browser.tcl (see [Example 9-1 on page 9-4](#)) creates a menu that looks like the following:



Clicking the Browse button next to Library Name opens the following dialog box:



Example 9-1 Creating a Library and Cell Name Browser – browser.tcl

```
# Code for browser.tcl
# Proc for library name browser
proc Browse {dir} {
global NewLibName
set NewLibName [tk_chooseDirectory -title \
"select Library" -initialdir $dir]
}

#Proc for cellname browser
proc cellBrowse {cell} {
global CellName
set CellName [tk_getOpenFile -title "select cell" -
```

```

initialdir $cell]
}

#Create buttons and frames for browser
# define the main window
set w .myMenu
toplevel $w -borderwidth 2
wm title $w "Milkyway Menu"
wm iconname $w "Menu"

# setup the frame for OK and CANCEL
frame $w.frame1 -borderwidth 5 -relief groove
button $w.frame1.ok -text "OK" -command myOk
button $w.frame1.cancel -text "Cancel" -command myCancel
pack $w.frame1.ok $w.frame1.cancel -side left -in $w.frame1

# setup frame for Library name and cell name
frame $w.frame2 -borderwidth 5 -relief groove

# Get Library Name
frame $w.frame2.alib -borderwidth 5
label $w.frame2.alib.txt -text "Library Name" -padx 5 -width 20 -anchor e
entry $w.frame2.alib.e1 -width 40 -textvariable NewLibName
button $w.frame2.alib.b1 -text "Browse..." -command {Browse .}
pack $w.frame2.alib.txt -side left -anchor w
pack $w.frame2.alib.e1 -side left
pack $w.frame2.alib.b1 -side left

# Get Cell Name
frame $w.frame2.cel -borderwidth 5
label $w.frame2.cel.txt -text "Cell Name" -padx 5 -width 20 -anchor e
entry $w.frame2.cel.e1 -width 40 -textvariable CellName
button $w.frame2.cel.b1 -text "Browse..." -command {cellBrowse $NewLibName
$viewName}
pack $w.frame2.cel.txt -side left -anchor w
pack $w.frame2.cel.e1 -side left
pack $w.frame2.cel.b1 -side left
pack $w.frame2.alib -side top -fill x
pack $w.frame2.cel -side top -fill x

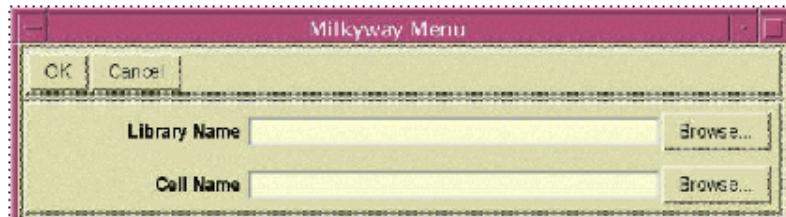
```

Tcl/Tk Scheme Interface

This section describes how to interface a Tcl/Tk program within the Milkyway Scheme environment. Any Tcl/Tk menu can be called in the Scheme environment, using the Scheme command `loadUserForm`. For example, to access the browser menu created in the previous section, use the Scheme command

```
loadUserForm "myMenu.tcl"
```

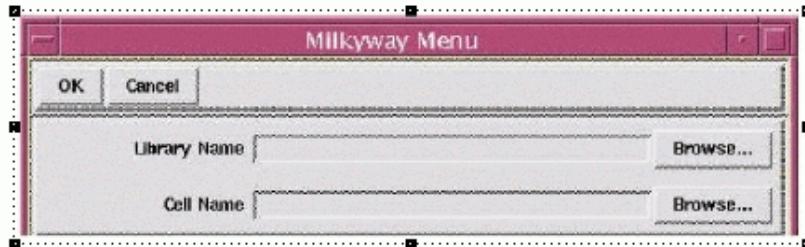
This displays the following dialog box:



This dialog box does not look like the default Milkyway menu. To make it look like a standard Milkyway dialog box, you set the following fonts and colors in your Tcl/Tk script:

```
set avnt_default_font *adobe*helvetica*bold-r-normal*12*
set avnt_menulabel_font *adobe*helvetica*bold-o-normal*12*
set avnt_entry_font *adobe*helvetica*medium-r-normal*12*
set avnt_text_font *adobe*courier*medium-r-normal*12*
set avnt_tag_font *adobe*courier*bold-r-normal*12*
option add *font $avnt_default_font
option add *background #d9d9d9
option add *foreground #000000
option add *activeBackground #ececfc
option add *activeForeground #000000
option add *selectBackground #c3c3c3
option add *disabledForeground #a3a3a3
option add *troughColor #c3c3c3
option add *highlightBackground #d9d9d9
```

By using these commands in the browser script, you make the menu look like the following:



Passing Variables to Scheme From Tcl/Tk

An important step in developing your Tcl/Tk GUI is passing variables from Tcl/Tk to the Milkyway Scheme environment. For example, the name of the library entered in the Tcl browser menu needs to be passed to a Scheme variable so that it can be used with other applications.

You pass values of variables from Tcl/Tk to Scheme by using the Tcl/Tk command `userEvalScm`. This command takes a Scheme command as an argument. The syntax is

```
userEvalScm "<scheme command>"
```

For example,

```
userEvalScm "geOpenLib \"$libName\""
```

If the library exists, the Tcl/Tk command in this example opens a library with the name defined by the Tcl variable `$libName`.

Creating Menus in Milkyway

This section explains how to create menus by using Scheme and how to customize the menu bars in the Milkyway GUI.

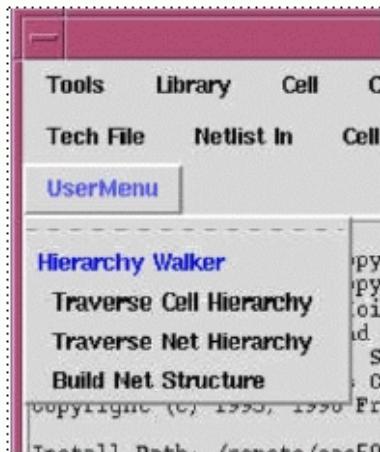
Creating Menus With Scheme

The Scheme command to add a user menu is `addUserMenu`. To remove a user menu use the `removeUserMenu` command.

When you use Scheme to create a menu in the Milkyway GUI, there are some limitations:

- Only drop-down menus can be created using Scheme.
- Each menu item must be associated with a Scheme function.

The Scheme code in [Example 9-2 on page 9-8](#) creates a menu with three items in the Milkyway GUI, as shown here below:



Example 9-2 Creating a User Menu With Scheme

```
(addUserMenu "UserMenu" (
```

```
list "Hierarchy Walker"
  (cons " Traverse Cell Hierarchy" "cellHier")
  (cons " Traverse Net Hierarchy" "netHier")
  (cons " Build Net Structure" "buildNetStruct")
))
```

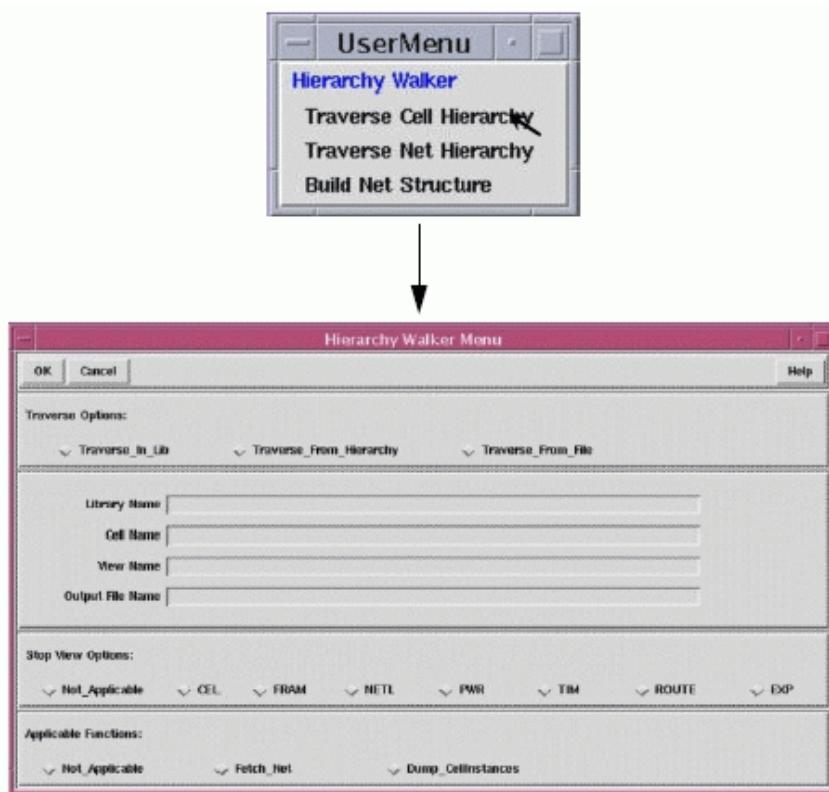
In [Example 9-2](#) each menu item is associated with a Scheme function. The Scheme function can be a procedure, which calls a Tcl GUI. A click on the menu entry then starts the Tcl/Tk GUI.

The code in [Example 9-3](#) defines a Scheme procedure called `cellhier`, which opens a dialog box when selected. [Figure 9-4](#) shows the menu and dialog box navigation.

Example 9-3 Creating a Scheme Procedure

```
(define cellHier
  (lambda ()
    (begin
      (loadUserForm "cellhier.tcl")
    )
  ))
```

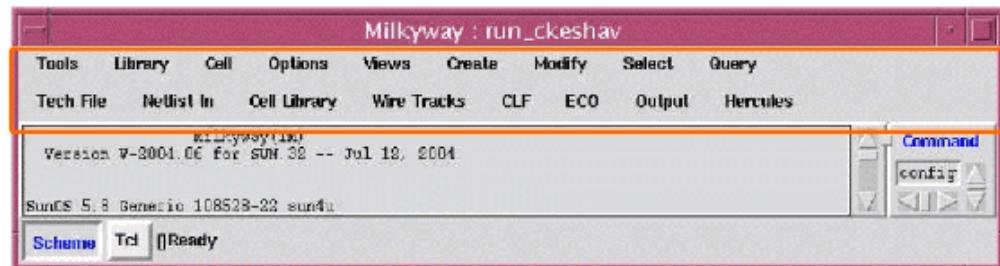
Figure 9-4 Calling a Dialog Box from a Menu



About the Default Milkyway Tools Menu

This section discusses the Milkyway Tools menu (see [Figure 9-5](#)). Do not modify any code in the following examples. Instead, use the menu as an example for developing your own tools menu, which is covered in detail in [“Adding a New Menu to the Milkyway GUI” on page 9-13](#).

Figure 9-5 Milkyway Tools Menu



The default Milkyway menus are installed in the etc/menus directory of your installation path. Each file in this directory defines a menu in Milkyway. Menu files must have the extension .menu (for example, Milkyway.menu).

[Example 9-4](#) shows the menu file Milkyway.menu for the Milkyway GUI shown in [Figure 9-5](#).

Example 9-4 Milkyway.menu

```
MENU_Bar 0
MENU_Group "Tools" 0
MENU_Button "Quit" menuQuit
MENU_GroupEnd

MENU_File "libcell_mw"
MENU_File "options_mw"
MENU_File "views_mw"
MENU_File "edit_mw"
MENU_File "select_mw"
MENU_File "query_mw"
MENU_Bar 1
MENU_File "techfile_mw"
MENU_File "dataprep_mw"
MENU_File "output_mw"
MENU_File "hercules
```

The content of the Milkyway.menu file is explained in detail below.

```
MENU_Group "Tools" 0
```

Specifies the text label "Tools" for the menu bar.

```
MENU_Button      "Quit"      menuQuit
```

Specifies the text label "Quit" and the Scheme command `menuQuit` for the menu button.

```
MENU_GroupEnd
```

Ends the definition of the Tools menu.

```
MENU_File "libcell_mw"
```

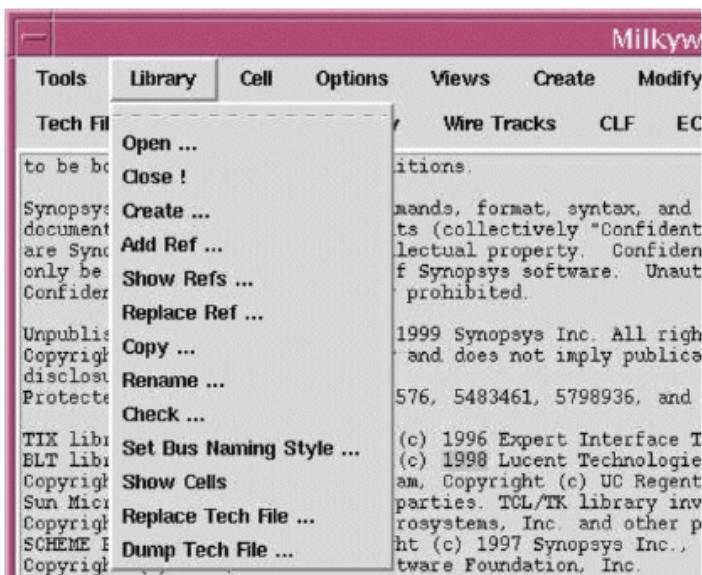
Calls the `libcell_mw.menu` menu file.

```
MENU_Bar -
```

Creates a new line.

[Figure 9-6](#) shows the Library menu followed by its menu file source code ([Example 9-5](#)), which is stored in the `/etc/menus` directory.

Figure 9-6 Library Menu GUI



Example 9-5 Library Menu Source Code

```
MENU_Group      "Library" 0
MENU_Button    "Open ..."     geOpenLib
MENU_Button    "Close !"      geConfirmCloseLib
MENU_Button    "Create ..."   cmCreateLib
MENU_Button    "Add Ref ..."  cmRefLib

.
.

MENU_GroupEnd
MENU_Group      "Cell" 0
MENU_Button    "Open ..."     geOpenCell
MENU_Button    "Close ..."   geCloseWindow

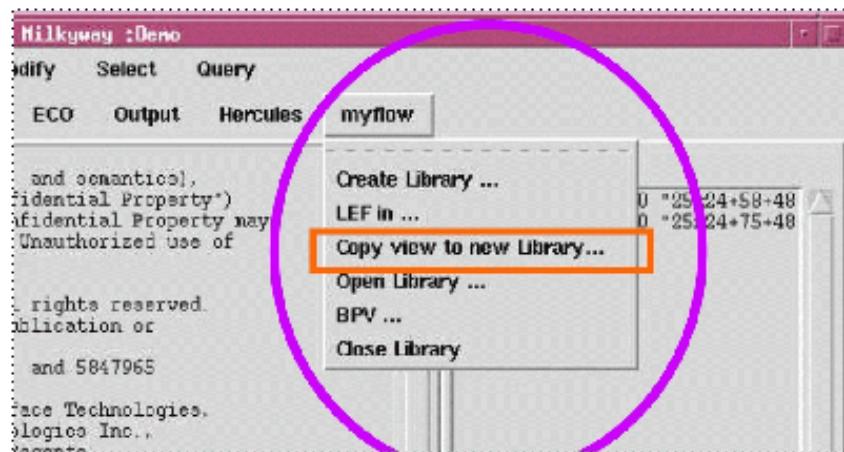
.
.

MENU_GroupEnd
```

Adding a New Menu to the Milkyway GUI

You can add your own menus to the Milkyway GUI by creating an appropriate menu file in the /etc/menus directory. [Figure 9-7](#) shows a new menu called myFlow that has been added to the Milkyway GUI.

Figure 9-7 Myflow Menu



To add this menu, you must first create the menu file (`myFlow.menu`) in `etc/menus`. The code to create the `myFlow` menu is shown in [Example 9-6](#).

Example 9-6 etc/menus/myFlow.menu

```
MENU_Group      "myflow"          0
MENU_Button     "Create Library ..." cmCreateLib
MENU_Button     "LEF in ..."      auNLIApi
MENU_Button     "Copy view to new view ..." copyView
MENU_Button     "Open Library ..."  geOpenLib
MENU_Button     "BPV ..."          auExtractBlockagePinVia
MENU_Button     "Close Library"   geConfirmCloseLib
MENU_GroupEnd
```

For the Milkyway GUI to recognize the new menu, you need to add the following line to the `Milkyway.menu`.

```
MENU_file "myflow"
```

In [Example 9-6](#), you might have noticed that the `copyView` command is not a Milkyway command. Therefore a Scheme procedure (`proc`) needs to be created in the `.avntrc` file for this menu to work properly. The code in [Example 9-7](#) shows how to define the Scheme command `copyView` as the Tcl command `copyview.tcl`.

Example 9-7 Defining copyView in .avntrc

```
(define copyView
  (lambda ()
    (begin
      (load "copyview.tcl")
    )))
```

This example shows how you can define a function and integrate it in a Milkyway menu. [Example 9-8](#) shows the complete myMenu.tcl program.

Example 9-8 Full Script of myMenu.tcl

```
#  
# define the main window  
#  
set w .myMenu  
toplevel $w -borderwidth 2  
wm title $w "Milkyway Menu"  
wm iconname $w "Menu"  
  
#  
#  
#Proc to get the browser for library  
#  
proc Browse {dir} {  
global NewLibName  
set NewLibName [tk_chooseDirectory -title "select Library" -initialdir $dir]  
if {[file exists $NewLibName/lib] == 0} {  
Browse $NewLibName  
}  
}  
#  
#  
#Procedure for cell name browser  
#  
proc cellBrowse {cell} {  
global CellName  
set Cell1 [tk_getOpenFile -title "select cell" -initialdir $cell]  
set CellName [lindex [split [lindex [split $Cell1 /] end] :] 0]  
}  
  
#  
#  
#Procedure for view select  
#  
proc myView {libName} {  
global views  
set views [exec ls -F $libName | grep /]  
global l  
set l .list  
toplevel $l -borderwidth 2  
wm title $l "Select View"
```

```

frame $l.but
button $l.but.b -text "Hide" -command {userEvalScm "userFormCancel \"\$l\""}
pack $l.but.b -side left

listbox $l.lst -width 10 -height 7
foreach mem $views {
    $l.lst insert end [string range $mem 0 end-1]
}
pack $l.but
BlankLine $l b1
pack $l.lst

global ViewName
bind $l.lst <ButtonPress-1> {set ViewName [string range [lindex $views [\$l.lst index active]] 0 end-1]}
}

#
# Procedure BlankLine
#
proc BlankLine { parent args } {
    set f [frame $parent.$args]
    pack $f -side top -fill x
    label $f.txt -text " "
    pack $f.txt -side left -in $f
}

#
# Procedure myOK
#
proc myOk {} {
    global w
    global Cfile viewOpt funcOpt
    global OutputFileName NewLibName CellName ViewName
    userEvalScm "system \"mda -$Cfile $NewLibName $OutputFileName $funcOpt\""
}

#
# Procedure myCancel
#
proc myCancel {} {
    global w
    userEvalScm "userFormCancel \"\$w\""
}

```

```

}

#
# Procedure PutUpMenu
#

proc PutUpMenu {} {
    global w
    global OutputFileName CellName NewLibName
    global Cfile viewOpt funcOpt
#
#
#Commands to set the display features of the Milkyway menu
#
set avnt_default_font *adobe*helvetica*bold-r-normal*12*
set avnt_menuLabel_font *adobe*helvetica*bold-o-normal*12*
set avnt_entry_font *adobe*helvetica*medium-r-normal*12*
set avnt_text_font *adobe*courier*medium-r-normal*12*
set avnt_tag_font *adobe*courier*bold-r-normal*12*
option add *font $avnt_default_font
option add *background #d9d9d9
option add *foreground #000000
option add *activeBackground #ececfc
option add *activeForeground #000000
option add *selectBackground #c3c3c3
option add *disabledForeground #a3a3a3
option add *troughColor #c3c3c3
option add *highlightBackground #d9d9d9

#
# set up the menu
#
frame $w.frame1 -borderwidth 5 -relief groove
button $w.frame1.ok -text "OK" -command myOk
button $w.frame1.cancel -text "Cancel" -command myCancel
pack $w.frame1.ok $w.frame1.cancel -side left -in $w.frame1
BlankLine $w.frame1 b11
BlankLine $w.frame1 b12

#
# Filling Frame for input data from users
#
frame $w.frame2 -borderwidth 5 -relief groove

```

```

#
# Get Library Name
#
BlankLine $w.frame2 bl1
frame $w.frame2.alib -borderwidth 5
label $w.frame2.alib.txt -text "Library Name" -padx 5 -width 20 -anchor e
entry $w.frame2.alib.e1 -width 40 -textvariable NewLibName
button $w.frame2.alib.b1 -text "Browse..." -command {Browse .}
pack $w.frame2.alib.txt -side left -anchor w
pack $w.frame2.alib.e1 -side left
pack $w.frame2.alib.b1 -side left

#
# Get Cell Name
#
frame $w.frame2.cel -borderwidth 5
label $w.frame2.cel.txt -text "Cell Name" -padx 5 -width 20 -anchor e
entry $w.frame2.cel.e1 -width 40 -textvariable CellName
button $w.frame2.cel.b1 -text "Browse..." -command {cellBrowse $NewLibName/
$viewName}
pack $w.frame2.cel.txt -side left -anchor w
pack $w.frame2.cel.e1 -side left
pack $w.frame2.cel.b1 -side left

pack $w.frame2.alib -side top -fill x
pack $w.frame2.cel -side top -fill x
BlankLine $w.frame2 bl2

#
# Stop View Cell Options Frame
#
frame $w.frame3 -borderwidth 5 -relief groove
frame $w.frame3.text -borderwidth 5
label $w.frame3.text.txt -text "Select View: "
pack $w.frame3.text.txt -side left
BlankLine $w.frame3.text bl1

frame $w.frame3.opts -borderwidth 5

#Prepare first button for CEL view

radiobutton $w.frame3.opts.but -variable viewOpt \
    -text CEL -value CEL -width 4
pack $w.frame3.opts.but -side left -padx 10

```

```

#Prepare second button for FRAM view

radiobutton $w.frame3.opts.but1 -variable viewOpt \
    -text FRAM -value FRAM -width 5
pack $w.frame3.opts.but1 -side left -padx 14

#Prepare third button for NETL view

radiobutton $w.frame3.opts.but2 -variable viewOpt \
    -text NETL -value NETL -width 5
pack $w.frame3.opts.but2 -side left -padx 18

#Prepare fourth button for PWR view

radiobutton $w.frame3.opts.but3 -variable viewOpt \
    -text PWR -value PWR -width 4
pack $w.frame3.opts.but3 -side left -padx 20

#Prepare fifth button for TIM view

radiobutton $w.frame3.opts.but4 -variable viewOpt \
    -text TIM -value TIM -width 5
pack $w.frame3.opts.but4 -side left -padx 22

pack $w.frame3.text $w.frame3.opts -side top -fill x

#
# Pack the top-level frames
#
    pack $w.frame1 -side top -fill x
    pack $w.frame2 -side top -fill x
    pack $w.frame3 -side top -fill x
}

PutUpMenu

```

Attached-File Management

Attached-file management provides a flexible approach to saving information that is not defined in the Milkyway schema. Milkyway allows you to attach a file to any object in the database. However, in practice, you attach a file to libraries or more commonly to cells (designs).

This section covers attaching a file to a cell. The principles for attaching files to cells also apply to attaching files to libraries.

Attached files contain text or binary information specific to your flow. This information is understood by you and not Milkyway. Therefore you must manage the attached-file data. For example, Milkyway cannot manage byte ordering (endian) issues with binary attached files. Also, attached files are not opened or closed automatically with cells or libraries.

Tcl Attached File Command Overview

This section covers commonly used Tcl commands to attach, retrieve, check (status), and detach a file. A summary table of Scheme and Tcl commands for managing attached files is shown in [Table 9-1](#). In the following examples, the cell (current design) is called `vend_cell`.

To attach a file, use the `attach_file` command.

```
Milkyway> attach_file myfile.txt
Info:Attaches file default_0 to design 'vend_cell'.
1
```

The `myfile.txt` file is attached to the current design. The name `default_0` is its pseudo name, which Milkyway creates as a handle to `myfile.txt`.

Once the file is attached, you can open it by using the `get_attached_file` command.

```
Milkyway> get_attached_file default_0
/nfs/milkyway/user9/data/design1/CEL/vend_cell:1_7_tmp
```

To retrieve the attached file you must use the pseudo name that Milkyway assigns. Before you save your attached file, it is a temporary file. Milkyway appends a `_tmp` string after the attached-file name to indicate temporary status.

For example, the temporary file of `top:3_3` is named `top:3_3_tmp`. If the attached file is in a CEL view directory, this file naming convention designates the third attached file of a design named `top.CEL;3`.

To permanently save the cell with the attached file, use the `save_design` command.

```
Milkyway> save_design  
1
```

After saving, you can check the file's status by using the `get_attached_file` command.

```
Milkyway> get_attached_file default_0  
/nfs/milkyway/user9/data/design1/CEL/vend_cell:1_7
```

After the file is saved, it is finalized, and the `_tmp` suffix is removed. To detach the file from the cell, use the `detach_file` command.

```
Milkyway> detach_file default_0  
Info:Detaches file 'default_0' from design 'vend_cell'.  
1
```

Milkyway reports that your attached file, `default_0`, is detached from the cell `vend_cell`.

A summary of Tcl and Scheme commands for managing attached files is shown in [Table 9-1](#).

Table 9-1 Scheme and Tcl Command for Attached-File Management

Command mode	For a library	For a design
Scheme	<code>dbGetLibAttachedFile</code> <code>dbStoreLibAttachedFile</code> <code>dbExistsLibAttachedFile</code> <code>dbDeleteLibAttachedFile</code> <code>dbDumpLibAttachedFiles</code>	<code>dbGetCellAttachedFile</code> <code>dbStoreCellAttachedFile</code> <code>dbExistsCellAttachedFile</code> <code>dbDeleteCellAttachedFile</code> <code>dbDumpCellAttachedFile</code>
Tcl		<code>attach_file</code> <code>detach_file</code> <code>get_attached_file</code>

A restriction on using links rather than duplicating attached files when copying cells is that links cannot be made across different file systems as is possible with UNIX symbolic links. When a cell is copied across file systems, the attached file is not linked but duplicated.

Specifying Attached-File Link Mode

Milkyway tries to minimize storage whenever possible. Provided that the attached file is not modified, if you copy a cell in link mode, the attached file is not duplicated. Instead Milkyway maintains a link to the first (original) attached file. This file is reserved, and identical files are linked to it.

If you want Milkyway to set a link and you prefer not to duplicate the attached file when you copy a cell, use the `set_attach_file_mode` command.

To set attached-file mode to link mode (no duplication), enter the following:

```
Milkyway> set_attach_file_mode 1
```

To set the attached-file link mode to duplicate the attached file when you copy a cell, set the link mode to 0. The default is copy, not link. Specifically, link mode is 0 by default.

```
Milkyway> set_attach_file_mode 0
```

By setting the value to 0, you turn off linking to reduce attached-file disk space. Every time you copy a cell, the attached file is copied as well.

If you have not set the link mode with `set_attach_file_mode`, you can use this command to check the link status. Milkyway returns 1 for link mode and 0 for duplicate mode.

Removing Attached Files by Linking

Milkyway provides the `purge_attach_file` command to clean up attached files through linking. For specifics on syntax, see the man page or get help by entering

```
Milkyway> purge_attached_file -help
```

The `purge_attach_file` command allows you to reduce duplication by linking attached files to the first attached file. The syntax is:

```
purge_attached_file
  [-effort low | medium | high]
  [-files {<pseudoname> <design1> <design2> ... }]
  [-reset_index]
  library_name
```

This command provides three levels of cleanup effort: low, medium, and high. With low effort, identical attached files with the same pseudo name (type) that belong to different versions of the same design are linked. For example, if `top:1_3` and `top:2_3` are identical, Milkyway links them to remove duplication.

Medium effort removes the constraint of having the same design. For example, if `top:1_3` is identical to `new:2_3` (copied from the former), the two are linked to save space.

High effort removes the constraint of having the same design and same pseudo name. Using high effort, Milkyway attempts to link all attached files regardless of design or pseudo name. Of the three options, this option takes the most memory and runtime to perform.

Using the `purge_attach_file` command, you can also specify files to link to the original attached file to save storage. An attached file can be specified by its design name and pseudo name. By default, all specified files are linked to the original attached file.

Before linking, Milkyway performs a complete comparison. If the file contents are different, Milkyway does not perform the link. However, you can force the link by using the `-force` option. Milkyway then ignores the differences, and all files are linked to the original.

With the `-force` option, you can specify only one type and two or more design names. Be careful when using this option, because data can be lost if the files are different. The following is an example:

```
Milkyway> purge_attach_file my_lib -files  
          {mypseudoname_1 top.CEL;1 top.CEL;2} -force}
```

The `purge_attach_file` also provides an option to reset the index. For example, you have a design that for some reason has two attached files named `top:3_2` and `top:3_5`, and Milkyway names the next attached file `top:3_6`. After the index is reset, your attached files are `top:3_1` and `top:3_2`. The next attached file is `top:3_3`.

The `purge_attach_file` execution speed depends on the size of the attached files and the frequency of their duplication. After linking, the attached file might be modified unexpectedly. As a precaution, you should

- Dump the attached file to a temporary file

- Detach the attached file
- Modify the temporary file, then attach it back

A

Milkyway Database Records

This appendix describes the records comprising the Milkyway database.

Common Fields

This section describes the most common fields in database records. Many of these bit values have a character mnemonic available through the function “[CONSTANT](#)” on page 3-3. The CONSTANT value is preferred for future upward compatibility, and code readability.

accessDirection

The *accessDirection* field indicates the access direction for pins, as follows:

- Bits 0 through 3 (CONSTANT “*dbbAccessDirMask*”) store information corresponding to the access direction of the pin.
- Bit 0 (CONSTANT “*dbbRightAccessDir*”) set to 1 indicates right access.
- Bit 1 (CONSTANT “*dbbLeftAccessDir*”) set to 1 indicates left access.
- Bit 2 (CONSTANT “*dbbUpAccessDir*”) set to 1 indicates top access.
- Bit 3 (CONSTANT “*dbbDownAccessDir*”) set to 1 indicates bottom access.
- Bits 4 through 7 store information from corresponding port direction. The port or region port sets these bits. The database maintains consistency between the port and associated pins.
- Bit 4 (CONSTANT “*dbbPinInputPort*”) set to 1 specifies the pin as an input port.

- Bit 5 (CONSTANT “*dbbPinOutputPort*”) set to 1 specifies the pin as an output port.
 - Bit 6 (CONSTANT “*dbbPinInputOutputPort*”) set to 1 specifies the pin as an input/output port.
 - Bit 7 (CONSTANT “*dbbPinTriState*”) set to 1 specifies the pin as a tristate port.
-

allowablePattern

Note:

Only AstroGA uses the allowablePattern field.

Allowable patterns are the legal orientations of the cell during placement. An integer, which is converted to binary, indicates how the cells are oriented when placed on the tile.

The following table shows the legal orientations. The Fs illustrate these orientations.

Table A-2

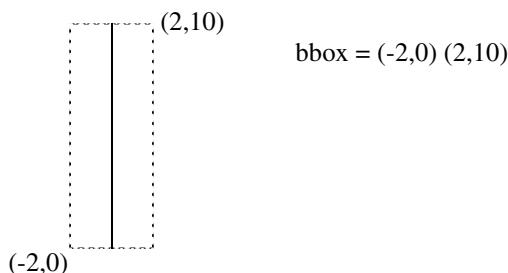
If the following bit is set:	The cell can be placed with the following orientation:	As shown below:
0	No transformation	F
1	Flipped on the y axis	⋮
2	Flipped on the x axis	⠄
3	Rotated 180 degrees	⠄⋮

Table A-2

If the following bit is set:	The cell can be placed with the following orientation:	As shown below:
4	Flipped on the x axis and rotated 90 degrees	□□
5	Rotated 90 degrees	□
6	Rotated 270 degrees	□
7	Flipped on the y axis and rotated 90 degrees	□□

bbox

Many records containing physical information about the design have a *bbox* field. The *bbox* field describes the bounding box (physical extent) of the object as a pair of x,y coordinates. For example, a path's bounding box encloses the coordinates of the path.



The first coordinate in the *bbox* field is the lower left corner of the box. The second coordinate is the upper right corner of the box. Internally, the database stores the box in database units. The Scheme interface converts the database units to user units.

cellFlag

Cell related records have a field called *cellFlag*. This field identifies the cell type and cell sub-type of the cell. The following table defines the *cellFlag* bit definitions.

Table A-3

bit	CONSTANT equivalent:	Interpretation
3	“dbbMaskLayout ViewType”	The cell contains graphic mask data. It may or may not have netlist information.
4	“dbbNetListView Type”	The cell is a netlist only cell. Generally the result of an edif or verilog translation. No graphics will be present.
5	“dbbSchematic ViewType”	Not Used
6	“dbbSymbolic ViewType”	Not Used
7	“dbbPCBLayout ViewType”	Not Used
8	“dbbArcPlanCell”	Cell is in the floorplanning phase
9	“dbbLogicModel ViewType”	Cell contains timing information. Usually has a view name of “.TIM”
10	“dbbArcChipCell”	Cell was generated by Astro, and is a fixed die standard cell design.
11	“dbbArcCellCell”	Cell was generated by Astro, and is a variable die standard cell design.
12	“dbbGalaxyCell”	Cell is a gate array design.

Table A-3

bit :	CONSTANT equivalent:	Interpretation
-	“dbbLayoutCell SubTypeMask”	The following bits are used to further differentiate cells of type <i>dbbMaskLayoutViewType</i> . This constant value can be used to extract the subtype from a mask layout cell.
13	“dbbIOPadCell”	Cell is used as a pad cell.
14	“ <i>dbbStandardCell</i> ” or “ <i>dbbModuleCell</i> ”	Cell is used as a Standard Cell. Module cell is the equivalent term for Gate Array design.
15	“dbbCornerPadCell”	Cell is used specifically in the corner of the design.
16	“ <i>dbbMacroCell</i> ”	Cell is used as a sub-block to a chip cell or as a sub-block of another cell.
17	“dbbChipCell”	Cell is a top level design. It will have bonding pads.
18	“dbbImageCell”	Used by AstroGA/Astro. The underlying base array info cell.
19	“ <i>dbbDoubleHeightSt dCell</i> ” or “ <i>dbbCoverCell</i> ”	For Astro, indicates the standard cell is “double high”. For AstroGA/Astro, indicates the cell is a cover cell.
20	“dbbFillerCell”	Used in conjunction with dbbStandardCell and dbbIOPadCell to indicate the cell is used for filling well etc.
30	“dbbSpecialVia Cell”	Cell is used as a special via in AstroGA.
-	“dbbLModelCell SubTypeMask”	The following bits are used to further differentiate cells of type <i>dbbLogicModelViewType</i> . This constant value can be used to extract the subtype from a logic cell.

Table A-3

bit :	CONSTANT equivalent:	Interpretation
21	“dbbFlipFlopCell”	Identifies the cell as a flip-flop or register.
22	“dbbClockBuffer Cell”	Identifies the cell as a clock buffering for use by the Clock Trunk Synthesis.
23	“dbbLatchCell”	Identifies the cell as a latch.
24	“dbbRamCell”	Identifies the cell as a RAM.
25	“dbbRomCell”	Identifies the cell as a ROM.
27	dbbFlipChipPadCell	Used in conjunction with <i>dbbIOPadCell</i> (bit 13) to identify a Flip-Chip pad.

cellRegionNumber

Each cell region has a *cellRegionNumber* containing an unique number from 0 to 255. If the *cellRegionNumber* is zero (0), either the net is associated with inter-region routing or the net data has not been partitioned to the regions. If the region number is non-zero, the net is associated with the specified region.

direction

The *direction* field indicates the port direction, as follows:

- Bits 0 through 3 (CONSTANT “*dbbPortDirectionMask*”) specify the port direction.
- Bit 0 (CONSTANT “*dbbInputPort*”) set to 1 indicates that the port is an input port.

- Bit 1 (CONSTANT “*dbbOutputPort*”) set to 1 indicates that the port is an output port.
- Bit 2 (CONSTANT “*dbbInputOutputPort*”) set to 1 indicates that the port is an input/output port.
- Bit 3 (CONSTANT “*dbbTriState*”) set to 1 indicates that the port is a tristate port.
- Bits 4 through 7 (CONSTANT “*dbbPortStatusMask*”) indicate the following. (Note that the constant values have already been shifted left 4 bits to allow direct comparison.)
 - If equal to 0, the port is normal.
 - If equal to 1 (CONSTANT “*dbcPortIsTieHigh*”), the port is auto tie-high.
 - If equal to 2 (CONSTANT “*dbcPortIsTieLow*”), the port is auto tie-low.
 - If equal to 3 (CONSTANT “*dbcPortIsVDD*”), the port is power.
 - If equal to 4 (CONSTANT “*dbcPortIsGND*”), the port is ground.

flag

This field internal to Synopsys applications.

In Astro, some records use the *flag* field to specify what type of operation created the object. These values can be accessed via the CONSTANT function. Valid constant string values include:
dbcPGRouteType, *dbcClockRouteType*,
dbcSignalGlobalRouteType, *dbcSignalUserDetailRouteType*,
dbcPGRingRouteType, *dbcPGStripeRouteType*,
dbcPGPinConnectRouteType, *dbcPGFollowPinRouteType*,
dbcClockRingRouteType, *dbcClockStripeRouteType*.

layer

A *layer* field contains an integer that defines the design layer associated with a record.

Many physical records are associated with a layer. The database stores the layer as a word. The technology file and most of the user interface permit usage of layer names or numbers. The current Scheme interface only supports layer numbers.

localStatus

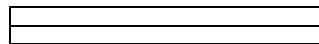
Port Instance and Pin Instance records contain a *localStatus* field, which allows the instance to have some unique information about them. The field is one byte wide, and are interpreted as follows:

- If bit 0 (CONSTANT “*dbbPortInstAutoTie*”) is set, it indicates the port is tied to either the instances power port or ground port. The router treats these connections special. They are typically pre-routed for area reduction.
- If bit 1 (CONSTANT “*dbbPortInstTieHigh*”) is set, it implies that the tie is to power. If it is not set, the tie was to ground. This bit is used in conjunction with bit 0.
- If bit 2 (CONSTANT “*dbbPortInstNotChild*”) is set, it implies that this connection is not a member of the net. Certain nets are not made members of the net for performance. These are limited to power and ground type nets.

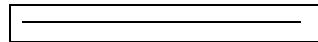
pathType

One of the following:

- If 0 (CONSTANT “*dbcSquareEnds*”), the path edge aligns with the coordinate.



- If 2 (CONSTANT “*dbcSquareEndsByHalfWidth*”), the path edge extends a half width beyond the coordinate.



- If 1 (CONSTANT “*dbcRoundEnds*”), the path edges are rounded.

Pin/Region Pin statusFlag

One of the following:

- Bit 0 (CONSTANT “*dbbPRPinPlaced*”) set to 1 places the pin.
- Bit 1 (CONSTANT “*dbbPRPinFixed*”) set to 1 fixes the pin.
- Bit 2 (CONSTANT “*dbbGRFeedThrough*”) set to 1 indicates a feedthrough.
- Bit 3 (CONSTANT “*dbbGRStraight*”) set to 1 indicates a straight pin (for pin type records).
- Bit 3 (CONSTANT “*dbbGRRRegionBus*”) set to 1 indicates a bus region pin (for region pin records).

- Bit 4 (CONSTANT “*dbbGRUser*”) set to 1 indicates a pin defined by the user.
- Bit 5 (CONSTANT “*dbbPinToPGRail*”) set to 1 indicates the pin is a PGRail pin.
- Bit 6 (CONSTANT “*dbbPinToStrip*”) set to 1 indicates the pin is a strip pin.
- Bit 7 (CONSTANT “*dbbRingPin*”) set to 1 indicates the pin is a ring pin.

routeType

routeType is internal to Synopsys applications.

The *routeType* field indicates the *routeType*, as follows:

Only bit (1-5) are used in the routing type definition.

- Major type - 2 bits
 - *axcSignalRouteType*
 - *axcPGRouteType*
 - *axcClockRouteType*
- Minor type - 3 bits
 - *axcSignalDetailRouteType*: Detailed Router
 - *axcSignalGlobalRouteType*: Global Router
 - *axcSignalUserDetailRouteType*: User-created, can be modified by Detailed Router(optional)
 - *axcPGRingRouteType*: PG Ring

- *axcPGStripeRouteType*: PG Stripe
- *axcPGPinConnectRouteType*: connect Macro-pin to ring/stripe
- *axcPGFollowPinRouteType*: Follow-pin
- *axcClockRingRouteType*: Clock Ring
- *axcClockStripeRouteType*: Clock Stripe
- *axcMiscBusRouteType*: Bus Routing
- *axcMiscFixedShieldRouteType*: fixed shielding
- *axcMiscDynamicShieldRouteType*: reroutable shielding

The default is *axcSignalDetailRoute*: Detailed Router

tile

Note:

Only AstroGA uses the tile field.

Tiles, the smallest unit of placement, are arranged in tile arrays in a repeating pattern that consists of four tiles. The four tiles are referred to by their positions as shown in the following figure.

UL	UR
LL	LR

Valid values:

- *UL* means the upper left corner.

- *LL* means the lower left corner.
 - *UR* means the upper right corner.
 - *LR* means the lower right corner.
-

transformCode

The *transformCode* field specifies the transformation of the cell instance with respect to the cell master.

The following table shows the legal orientations specified by the integers (converted to binary). The Fs illustrate these orientations. The CONSTANT equivalent is listed in parenthesis.

Table A-4

transform value:	The cell can be placed with the following orientation:	As shown below:
0	No transformation (“dbcRotate0”)	F
2	Rotated 90 degrees counterclockwise (“dbcRotate90”)	□
4	Rotated 180 degrees counterclockwise (“dbcRotate180”)	□
6	Rotated 270 degrees counterclockwise (“dbcRotate270”)	□
12	Flipped on the Y axis (“dbcRotate0MirrorX”)	□
10	Flipped on the X axis and rotated 90 degrees counterclockwise (“dbcRotate90MirrorX”)	□

Table A-4

transform value:	The cell can be placed with the following orientation:	As shown below:
8	Flipped on the X axis (“dbcRotate180MirrorX”)	⊤
14	Flipped on the Y axis and rotated 90 degrees counterclockwise (“dbcRotate270MirrorX”)	⊤⊤
8	Flipped on the X axis (“dbcRotate0MirrorY”)	⊤
14	Flipped on the Y axis and rotated 90 degrees counterclockwise (“dbcRotate90MirrorY”)	⊤⊤
12	Flipped on the Y axis (“dbcRotate180MirrorY”)	⊤
10	Flipped on the X axis and rotated 90 degrees counterclockwise (“dbcRotate270MirrorY”)	⊤⊤

Cell Hierarchy Records

CellInst Record (User View)

Not all information in the user view of a cell instance record is stored with each cell instance. The database automatically collects all necessary information when fetching a cell instance record. A cell instance record fetch is actually a double fetch, in that it fetches the cell instance record and the associated master record to create the user view record.

A cell instance record (user view) contains the fields described in the following sections.

bbox

See “[bbox](#)” on page A-4.

cellFlag

See “[cellFlag](#)” on page A-5.

cellInstMasterId

The *cellInstMasterId* field contains the database ID of the cell master linked to this *cellInst*.

flag2

The *flag2* file is internal to Synopsys applications.

flag3

The *flag3* field is internal to Synopsys applications.

flag1

The *flag1* field is internal to Synopsys applications. It contains a status byte for the cell instance. It is divided into two subfields:

- Bits 0 through 2 (CONSTANT “*dbbPRCellInstPlacementMask*”) specify the instance placement status:
 - Bit 0 (CONSTANT “*dbbPRCellInstPlaced*”) set to 1 places the cell instance.
 - Bit 1 (CONSTANT “*dbbPRCellInstFixed*”) set to 1 fixes the cell instance.
 - Bit 2 (CONSTANT “*dbbPRCellInstRigidFixed*”) set to 1 rigid fixes the cell instance.
- Bits 3 through 5 (CONSTANT “*dbbPRCellInstECOMask*”) specify the ECO status of the instance:

- Bit 3 (CONSTANT “*dbbPRCellInstECODelete*”) set to 1 indicates the placer will delete the cell instance during the ECO placement stage. The instance is not deleted until placement to allow a hint to the placer.
- Bit 4 (CONSTANT “*dbbPRCellInstECOAdd*”) set to 1 identifies this instance to the placer as a new instance. After the ECO placer places the cell, the bit is reset. If both bits 3 and 4 are set, it indicates to the placer that the cell instance has had a master change, and that cell row evening will be needed.
- Bit 5(CONSTANT “*dbbPRCellInstECOLgnore*”) set to 1 identifies the instance as an instance that should be ignored during an ECO netlist comparison. This indicates that this instance was interactively added by the user and should not be deleted even if it does not appear in the ECO netlist.

transformCode

See “[transformCode](#)” on page A-13.

name

The *name* field contains the database name associated with this cell. If the place and route netlist was flattened, a cross-reference table is available to translate flat names to hierarchical names.

position

The *position* field specifies the origin of this cell instance in (x, y) coordinates.

cellFlag

See “[cellFlag](#)” on page A-5.

CellInstArray Record

A CellInstArray record contains the fields described in the following sections.

descriptorLibName

The *descriptorLibName* field comprises a text string that references the library containing the *descriptor*, if not current.

oldName

The *oldname* field contains a user specified text string that refers to original name of this *CellInstArray* record.

flag1

The *flag1* field is internal to Synopsys applications.

flag2

The *flag2* field is internal to Synopsys applications.

flag3

The *flag3* field is internal to Synopsys applications.

transformCode

See “[transformCode](#)” on page A-13.

bbox

See “[bbox](#)” on page A-4.

cellName

The *cellNam* field contains the cell name of the cell master.

viewName

The *viewName* field contains the view name of the cell master.

cellVersion

The *cellVersion* field contains an integer version of the cell instance.

position

The *position* field specifies the origin of the cell instance in array coordinates.

cellFlag

See “[cellFlag](#)” on page A-5.

cellInstMasterId

The *cellInstMasterId* of the cell instance record that this cell instance array record refers to.

xDisplacement

The *xDisplacement* field contains the distance between cell instances in the horizontal direction.

yDisplacement

The *yDisplacement* field contains the distance between cell instances in the vertical direction.

XduplicateN

The *XduplicateN* field indicates the number of cell instances in the horizontal direction.

YduplicateN

The *YduplicateN* field indicates the number of cell instances in the vertical direction.

name

The *name* field contains the user specified text string identifying this record.

CellInstMaster Record

The CellInstMaster record facilitates database speed and minimizes database size. The master provides most of the cell header information within the top cell, eliminating the need to open the macro or standard cell for each redraw. The database keeps only one copy of each cell type, regardless of how many instances exist within the cell.

A cell instance master record contains the fields described in the following sections.

cellBoundaryId

The *cellBoundaryId* is the object ID of the *cellBoundary* within the Boundary record to which this CellInstMaster record refers.

describorLibName

The *describorLibName* field contains a text string that refers to the library that contains the describor, if not current.

flag

The flag field is internal to Synopsys applications. See “[flag](#)” on page A-8.

name

The *name* field contains the user specified text string identifying this record.

prBoundaryId

The *prBoundaryId* is the *prBoundary* object ID in the Boundary record that this CellInstMaster record refers to.

propertyId

This field is currently designated as internal to Synopsys applications.

cellBoundaryBBox

The *cellBoundaryBBox* field contains the bbox of the cell boundary. See “bbox” on page A-4.

prBoundaryBBox

The *prBoundaryBBox* field contains the bbox of the pr boundary. See “bbox” on page A-4.

type

The type of the cell that cellinstmaster references.

logicalEquivalentClass

The *logicalEquivalentClass* field contains the *logicalEquivalentClass* of the cell. If the *logicalEquivalentClass* field is set to 0, no other cells are logically equivalent to this cell. If this field is non-zero, all other cells with the same *logicalEquivalentClass* value are logically equivalent to this cell.

cellClass

The *cellClass* field is an 8-bit field that indicates the cell classes that are allowed in a row. If this field is 0, any cell is allowed. If the *cellClass* is non-zero, only cells with the same *cellClass* are allowed.

numberOfCellInst

The *numberOfCellInst* field contains the number of cell instances that reference this cell master.

bbox

See “[bbox](#)” on page A-4.

cellName

The *cellName* field contains the cell name of the cell master.

viewname

The *viewName* field contains the view name of the cell master.

cellVersion

The *cellVersion* field contains an integer version of this cell instance.

cellFlag

See “[cellFlag](#)” on page A-5 for definition.

numberOfPin

The *numberOfPin* field specifies the number of pins in this cell instance master.

numberOfPort

The *numberOfPort* field specifies the number of ports in this cell instance master.

ExpCellInst Record

An *ExpCellInst* record contains the fields described in the following sections.

bbox

See “[bbox](#)” on page A-4.

cellBoundary

This field is currently designated as internal to Synopsys applications.

cellBoundaryBBox

The *cellBoundaryBBox* field contains the bbox of the cell boundary. See “[bbox](#)” on page A-4.

cellClass

This field is currently designated as internal to Synopsys applications.

cellName

The *cellName* field contains a text string that refers to the cell name of this expanded cell instance.

cellVersion

The *cellVersion* field contains an integer version of the expanded cell instance.

inst

inst comprises the following fields:

- name

The *name* field contains the user specified text string identifying this record.

- position

The *position* field specifies the cell instance origin om array coordinates.

- transformCode

See “[transformCode](#)” on page A-13.

- flag1

The flag1 field is internal to Synopsys applications.

- flag2

The flag2 field is internal to Synopsys applications.

- flag3

The flag3 field is internal to Synopsys applications.

- cellFlag

See “[cellFlag](#)” on page A-5 for definition.

modifyDate

The *modifyDate* field contains an unsigned integer associated with the cell or library modify date. The *modifyDate* includes day, month, year, sec, minute, and hour.

numberOfPin

The *numberOfPin* field specifies the number of pins in this expanded cell instance.

numberOfPort

The *numberOfPort* field specifies the number of ports in this expanded cell instance.

prBoundary

This field is currently designated as internal to Synopsys applications.

prBoundaryBBox

The *prBoundaryBBox* field contains the *prBoundary* bbox. See “[bbox](#)” on page A-4.

statusFlag

The *statusFlag* field is internal to Synopsys applications. See “[Pin/Region Pin statusFlag](#)” on page A-10“for definition.

viewName

The *viewName* field contains the view name of the cell master.

ExpPinInst Record

accessDirection

See “[accessDirection](#)” on page A-2.

cellInstId

The *cellInstId* field specifies the object ID of the cell instance to which this *ExpPinInst* record is attached.

childNetId

The *childNetId* field contains the object ID of the child cell net to which this *ExpPinInst* record is attached.

childPinId

The *childPinId* field contains the object ID of the pin instance within the child cell to which this *ExpPinInst* record is attached.

childPortId

The *childPortId* field contains the object ID of the port instance within the child cell to which this *ExpPinInst* record is attached.

electricalEquivalentClass

The *electricalEquivalentClass* field inherits data from *PinInstMaster* record. See “[PinInstMaster Record](#)” on page A-28 for more information.

localStatus

See “[localStatus](#)” on page A-9.

mustJoinClass

This field is currently designated as internal to Synopsys applications.

name

The *name* field contains the user specified text string identifying this record.

netId

The *netId* field contains the object ID for the net to which the *ExpPinInst* record is attached.

pinInstMasterId

The *pinInstMasterId* field contains the pin instance master database ID for this *ExpPinInst* record.

portInstId

The *portInstId* field contains the object ID of the *portInst* that the *ExpPinInst* references.

bbox

See “[bbox](#)” on page [A-4](#) for definition.

layer

See “[layer](#)” on page [A-9](#) for definition.

numberOfPoint

The *numberOfPoint* field contains the number of points that define the *ExpPinInst* record.

points

The *points* field contains the points that graphically define the *ExpPinInst* record.

statusFlag

The *statusFlag* field is internal to Synopsys applications. See “[Pin/Region Pin statusFlag](#)” on page [A-10](#) for definition.

statusFlag1

The *statusFlag1* field is internal to Synopsys applications.

ExpPortInst Record**childNetId**

The *childNetId* field contains the object ID of the child cell net to which this *ExpPortInst* record is attached.

childPortId

The *childPortId* field contains the object ID of the port instance within the child cell to which this *ExpPortInst* record is attached.

direction

See “[direction](#)” on page A-7 for definition

name

The *name* field contains the user specified text string identifying this record.

numberOfPinInst

The *numberOfPoint* field contains the count of the number of points that define the *ExpPortInst* record.

portInst

portInst comprises the following fields:

- *netId*

The *netId* field contains the object ID for the net to which this *ExpPortInst* record is attached.

- *cellInstId*

The *cellInstId* field contains the object ID of the cell instance that is associated with this *ExpPortInst* record.

- *localStatus*

See “[localStatus](#)” on page A-9.

- *constraints*

PinInstMaster Record

A *PinInstanceMaster* record contains the fields described in the following sections.

mustJoinClass

This field is currently designated as internal to Synopsys applications.

portInstMaster

The *portInstMaster* field contains the database ID of the *PortInstMaster* linked to *PinInstMaster*.

statusFlag

The *statusFlag* field is internal to Synopsys applications. See “[Pin/Region Pin statusFlag](#)” on page [A-10](#) for definition.

accessDirection

See “[accessDirection](#)” on page [A-2](#).

statusFlag1

The *statusFlag1* field is internal to Synopsys applications. One of the following may be set indicating:

- Bit 0 (CONSTANT “*dbbPinFollowInRow*”) set to 1 indicates the pin should follow the row.
- Bit 1 (CONSTANT “*dbbPinFollowInCol*”) set to 1 indicates the pin should follow the column.

electricalEquivalentClass

- The *electricalEquivalentClass* field is a byte field that indicates the electricalEquivalentClass and the *mustJoinClass* of the pin. This field is split as follows:
 - Bit0 through Bit15 are used as *electricalEquivalentClass*. The Synopsys application can have at most 65535 different *electricalEquivalentClasses*. Pins with the same *electricalEquivalentClass* are electrically equivalent.
 - Bit16 through Bit31 are used as must-join. The Synopsys application can have at most 65535 different *mustJoinClasses*. The router externally connects all pins on the same port with the same must-join numbers. There is no ordering constraint.

For example, pin one(p1), pin two (p2) and pin three (p3) have the same must-join class number, 1. The router connects pins one through three since they have the same must-join class number. All the following connections are acceptable:

p1 <-> p2 <-> p3

p1 <-> p3 <-> p2

p1 <-> some other connected net <-> p2 <-> p3

The default is zero *electricalEquivalentClasses* and zero must-join classes.

cellInstanceId

The *cellInstanceId* field contains the database ID of the cell master linked to this pin instance master.

bbox

See “[bbox](#)” on page A-4.

childPinId

The *childPinId* field contains the object ID of the pin instance inside the child cell to which this *PinInstMaster* record is attached.

childNetId

The *childNetId* field contains the object ID of the child cell net to which this *PinInstMaster* record is attached.

childPortId

The *childPortId* field contains the object ID of the port instance within the child cell to which this *PinInstMaster* record is attached.

name

The *name* field contains a text string that indicates the name of the pin associated with the pin instance.

layer

See “[layer](#)” on page A-9.

numberOfPoint

The *numberOfPoint* field contains the number of points in the *point* field.

points

The *points* field contains the (x,y) coordinate *rectArray* points associated with the *PinInstMaster* record. Because this The coordinates will not be transformed as this is a master record.

PortInst Record

Each time the database creates a cell instance, it also creates port instances that indicate the connection between the cell instance and the nets in the top cell. The *PortInst* record references a port instance master record where additional data about the port instance is stored. This link facilitates faster database speed and minimizes database size.

The *PortInstMaster* record provides most of the port header information within the top cell, eliminating the need to open the macro or standard cell for each operation. The database keeps only one copy of each port type, regardless of how many instances exist within the cell.

A *PortInst* record contains the fields described in the following sections.

constraints

This field is currently designated as internal to Synopsys applications.

localStatus

See “[localStatus](#)” on page A-9.

portInstMasterId

The *portInstMasterId* field contains the object ID for the *portInstMaster* attached to this *portInst* record.

netId

The *netId* field contains the object ID for the net to which this *PortInst* record is attached.

cellInstId

The *cellInstId* field contains the object ID of the cell instance that is associated with this port instance.

localStatus

See “[localStatus](#)” on page A-9.

PortInstMaster Record

The *PortInstMaster* record provides most of the port header information within the top cell, eliminating the need to open the macro or standard cell for each operation. The database keeps only one copy of each port type, regardless of how many port instances occur within the cell.

A *PortInstMaster* record contains the fields described in the following sections.

flag

The flag field is internal to Synopsys applications. See “[flag](#)” on page A-8 for definition.

direction

See “[direction](#)” on page A-7.

numberOfPinInstMaster

The *numberOfPinInstMaster* field contains the number of pins on master cells that are associated with this port instance.

cellInstMasterId

The *cellInstMasterId* field contains the database ID of the cell master linked to this port instance master.

portId

The *portId* field contains the object ID of the port inside the child cell that the port instance ID references.

netId

The *netId* field contains the object ID for the net inside the child cell that the port instance ID references.

name

The *name* field contains a text string identifying the port name that this record references.

Connectivity Records

Net Record

A Net record contains the fields described in the following sections.

constraints

This field is currently designated as internal to Synopsys applications.

numberOfOtherChild

The *numberOfOtherChild* field contains an integer that identifies the total number of other objects attached to this net.

parentId

This field is currently designated as internal to Synopsys applications.

powerId

This field is currently designated as internal to Synopsys applications.

propertyId

This field is currently designated as internal to Synopsys applications.

clientId

The *clientId* field contains the object ID of the client object that refers to this net

index

This field is currently designated as internal to Synopsys applications.

numberOfPort

The *numberOfPort* field contains the number of port attaches to this net.

numberOfPortInst

The *numberOfPortInst* field contains the number of *portInst* attached to this net.

bbox

See “[bbox](#)” on page A-4 for definition.

type

The *type* field contains a binary number that indicates the net type.

- Bit 0 (CONSTANT “*dbbSignalNet*”) set to 1 indicates a signal net.
- Bit 1 (CONSTANT “*dbbPowerNet*”) set to 1 indicates a power net.
- Bit 2 (CONSTANT “*dbbGroundNet*”) set to 1 indicates a ground net.
- Bit 3 (CONSTANT “*dbbClockNet*”) set to 1 indicates a clock net.
- Bit 4 (CONSTANT “*dbbSplitClockNet*”) set to 1 indicates a split clock net. Split clock nets are usually new nets created as the result of a clock tree synthesis.

flag

The *flag* field is internal to Synopsys applications. It is defined as one of the following:

- Bits 0-2 (CONSTANT “*dbbNetCriticalityMask*”) is a 8 valued field establishing the criticality of the net to the global router.
- Bit 4 (CONSTANT “*dbbNetSkipRoute*”) set to 1 indicates that the net should be skipped by the router.
- Bit 5 (CONSTANT “*dbbNetBitHasVariableWidth*”) set to 1 indicates that the net record should be routed in variable width.
- Bit 6 (CONSTANT “*dbbConsiderNetSrcSink*”) set to 1 indicates that the router should consider the source driver of the net when global routing the region.
- Bit 7 (CONSTANT “*dbbNetBitHasSinglePort*”) set to 1 indicates that the net has a single port.

cellRegionNumber

See “[cellRegionNumber](#)” on page A-7.

numberOfChildNet

The *numberOfChildNet* field contains an integer that identifies the total number of objects attached to this net.

name

The *name* field contains a text string that specifies the database name associated with this net. If the place and route netlist was flattened, a cross-reference table is available to translate flat names (the value of this field) to hierarchical names.

Pin Record

The Pin record supports the following types of pins:

- rectangle pins
- polygon pins
- pins with a list of connected rectangular shapes whose edges touch each other

A Pin record contains the fields described in the following sections.

dataType

This field is currently designated as internal to Synopsys applications.

firstSolution

This field is currently designated as internal to Synopsys applications.

firstWireSolution

This field is currently designated as internal to Synopsys applications.

mustJoinClass

statusFlag

The *statusFlag* field is internal to Synopsys applications. See “[Pin/Region Pin statusFlag](#)” on page A-10 for definition.

accessDirection

See “[accessDirection](#)” on page A-2.

statusFlag1

The *statusFlag1* field is internal to Synopsys applications. One of the following may be set indicating:

- Bit 0 (CONSTANT “*dbbPinFollowInRow*”) set to 1 indicates the pin should follow the row.
- Bit 1 (CONSTANT “*dbbPinFollowInCol*”) set to 1 indicates the pin should follow the column.

electricalEquivalentClass

The *electricalEquivalentClass* field is a byte field that indicates the *electricalEquivalentClass* and the *mustJoinClass* of the pin. This field is split as follows:

- The *electricalEquivalentClass* field is a byte field that indicates the *electricalEquivalentClass* and the *mustJoinClass* of the pin. This field is split as follows:

- Bit0 through Bit15 are used as *electricalEquivalentClass*. The Synopsys application can have at most 65535 different *electricalEquivalentClasses*. Pins with the same *electricalEquivalentClass* are electrically equivalent.
- Bit16 through Bit31 are used as must-join. The Synopsys application can have at most 65535 different *mustJoinClasses*. The router externally connects all pins on the same port with the same must-join numbers. There is no ordering constraint.

For example, pin one(p1), pin two (p2) and pin three (p3) have the same *mustJoinClass* number, 1. The router connects pins one through three since they have the same *mustJoinClass* number. All the following connections are acceptable:

- p1 <-> p2 <-> p3
- p1 <-> p3 <-> p2
- p1 <-> some other connected net <-> p2 <-> p3

The default is zero *electricalEquivalentClasses* and zero must-join classes.

bbox

See “[bbox](#)” on page A-4.

netId

The *netId* field contains the object ID that is a net object ID. This object ID is part of this net ID’s net.

portId

The *portId* field contains the object ID of the logical port of this pin.

name

The *name* field contains a text string indicating the name of the pin. Pin names are limited to 31 characters, and are unique within a given port.

layer

See “[layer](#)” on page A-9.

numberOfPoint

The absolute value of the *numberOfPoint* field is the size of the point array. The following conventions are enforced by the application:

- If this field is 0, the pin is a rectangle pin and the shape is stored in bbox.
- If this field is greater than 0, then the pin is a polygon pin.
- If this field is less than 0, the point field stores the first rectangle in point [0] and point [1], the second rectangle in point [2] and point [3] and so on.

points

One of the following:

- If positive, the pin is the same as other usages of the point field.
- If negative, the pin consists of $-n/2$ rectangles whose bounding boxes are stored in point.

Port Record

The Port record is associated with a net when the net is to be available outside the cell. Only one port can be associated with a net. The port may have multiple pins, but all pins are electrically connected inside the cell.

A Port record contains the fields described in the following sections.

mustJoinRootId

This field is currently designated as internal to Synopsys applications.

index

This field is currently designated as internal to Synopsys applications.

type

This field is currently designated as internal to Synopsys applications.

direction

See “[direction](#)” on page A-7.

numberOfPin

The *numberOfPin* field specifies the number of pins that are associated with this port.

netId

The *netId* field contains the object ID that is a net object ID. This object ID is part of this net ID’s net.

name

The *name* field contains the user specified text string identifying this record.

Database File Records

CellInfo Record

A CellInfo record contains the fields described in the following sections.

cellClockFrequency

This field is currently designated as internal to Synopsys applications.

cellPowerConsumption

This field is currently designated as internal to Synopsys applications.

cellVoltage

This field is currently designated as internal to Synopsys applications.

wireType

clientObj

This field is currently designated as internal to Synopsys applications.

creatorFlag

The *creatorFlag* field is internal to Synopsys applications.

flag

The *flag* field is internal to Synopsys applications.

name

The *name* field contains the user specified text string identifying this record.

prBoundaryObjId

The *prBoundaryObjId* field contains the object ID of the *prBoundary* object that is linked to the CellInfo record.

type

The *type* field contains one of the three cell types associated with a CellInfo record, including

- GenericCellType
- TieCellType
- RipperCellType

logicalEquivalentClass

The *logicalEquivalentClass* field specifies the *logicalEquivalentClass* of the cell. If 0, no other cells are logically equivalent to this cell. If non-zero, all other cells with the same *logicalEquivalentClass* value are logically equivalent to this cell.

schemaMajorVersion

The *schemaMajorVersion* field contains version information about the cell.

schemaMinorVersion

The *schemaMinorVersion* field contains version information about the cell.

cellFlag

See “[cellFlag](#)” on page A-5 for definition.

creator

The *creator* field contains a text string that specifies the user name of the process that created this cell.

creationDate

The *creationDate* field contains an unsigned integer that is associated with the creation date of the cell or library and comprises the year, month, day, hour, minute, and second that a cellInfo record is created.

lastWriter

The *lastWriter* field contains a text string that specifies the user name of the process that last modified the cell.

modifyDate

The *modifyDate* field contains an unsigned integer that is associated with the modification date of the cell or library.

allowablePattern

See “[allowablePattern](#)” on page A-3.

tile

See “[tile](#)” on page A-12.

cellClass

The *cellClass* field is an 8-bit field that indicates the cell classes that are allowed in a row. If 0, any cell is allowed. If non-zero, only cells with the same *cellClass* are allowed.

baseArrayClass

The *baseArrayClass* field contains a number that indicates the base array class that this cell row belongs to. The *baseArrayClass* of a cell row is defined inside the image cell. This field is used by AstroGA only.

imageCellName

The *imageCellName* field is currently not used. This field will be used by AstroGA only.

imageViewName

The *imageViewName* field is currently not used. This field will be used by AstroGA only.

imageViewVersion

The *imageViewVersion* field is currently not used. This field will be used by AstroGA only.

cellPowerValue

The *cellPowerValue* field contains the power consumed by the cell in database units. This field is used by AstroGA only.

subTileFlag

The *subTileFlag* is internal to Synopsys applications. It identifies the subtiles that are contained by the cell referenced by this record.

cellBoundaryObjId

The *cellBoundaryObjectId* field contains the object ID of the cell boundary. This is an internal field maintained by the application.

LibInfo Record

A *LibInfo* record contains the fields described in the following sections.

hierDelimiter

This field is currently designated as internal to Synopsys applications.

oTechHeader

The *oTechHeader* field is internal and not available for use by the user.

Error Cell Records

ErrorObj Record

An *ErrorObj* record contains the fields described in the following sections.

numberOfComponent

The *numberOfComponent* field contains a positive integer specifying the number of objects in the error.

bbox

See “[bbox](#)” on page A-4.

errorLevel

The *errorLevel* field is designated as internal to the Synopsys application.

errorTypeIid

The *errorTypeIid* field is designated as internal to Synopsys applications.

infold

The *infold* field contains a text string that indicates the message reported by the verification tool.

bbox

status

The *status* field is designated as internal to Synopsys applications.

ErrorType Record

An *ErrorType* record contains the fields described in the following sections.

level

One of the following:

- 0 gives an informational error.
- 1 gives a warning.
- 2 gives an error.
- 3 gives a severe error.

color

The *color* field contains an integer from 0 - 255 that specifies the color of the error.

pattern

The *pattern* field contains an integer from 0 - 255 that specifies the pattern of the error.

numberOfErrorObj

The *numberOfErrorObj* field contains a count of the number of errors detected for this error class.

bbox

See “[bbox](#)” on page A-4.

status

The *status* field is internal to the Synopsys application.

infold

The *infold* field contains a text string that specifies the class of error, for example, “poly-width” or “short”. The *infold* field is defined by LVS and DRC. This field is internal and unavailable for use by the user.

summary

The *summary* field contains a text string that indicates the message reported by the verification tool.

ErrorComponentRecord

The *ErrorComponentRecord* is internal to Synopsys applications. It contains the fields described in the following sections.

errorObjId

The *errorObjId* field is internal to Synopsys applications.

layer

The *layer* field is internal to Synopsys applications.

bbox

See “[bbox](#)” on page A-4.

netId

The *netId* field contains the object ID that is a net object ID. This object ID is part of this net ID’s net.

polygonNumber

The *polygonNumber* field is internal to the Synopsys application. This field contains an integer that is reserved for use by the verification tool.

nodeNumber

This field contains an integer that is reserved for use by the verification tool.

numberOfPoint

The *numberOfPoint* field contains the number of points in the *points* field.

points

The *points* field contains the points associated with an object.

Interregion Global Routing Records

GLink Record

A GLink record is used by Astro to pre-define some routing information.

dataType

This field is currently designated as internal to Synopsys applications.

wireMasterIdx

This field is currently designated as internal to Synopsys applications.

routeType

See “[routeType](#)” on page A-11.

layer

See “[layer](#)” on page A-9.

netId

The *netId* field contains the object ID for the net to which this GLink record is attached.

point1, point2

The start and end points of the jumper.

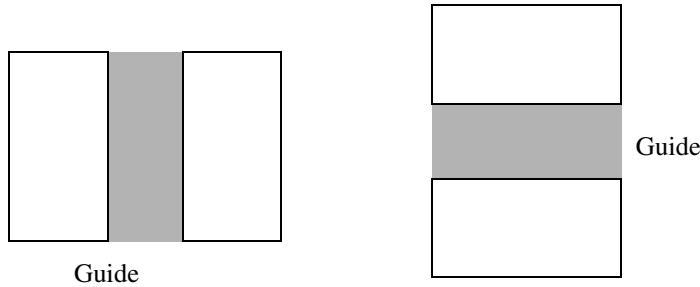
Guide Record

A Guide record contains the fields described in the following sections.

guideDirection

One of the following:

- 0 (CONSTANT “*dbcChannelXDir*”) specifies a horizontal direction.
- 1 (CONSTANT “*dbcChannelYDir*”) specifies a vertical direction.



status

One of the following:

- Bit 0 (CONSTANT “*dbbPreferredChannel*”) set to 1 indicates the guide is a preferred channel.
- Bit 1 (CONSTANT “*dbbObstacleChannel*”) set to 1 indicates the guide is an obstacle channel.
- Bit 2 (CONSTANT “*dbbObstacleTrunk*”) set to 1 indicates the guide is an obstacle trunk.

- Bit 3 (CONSTANT “*dbbSelectTrunkWidth*”) set to 1 indicates the guide is a select trunk width.
- Bit 4 (CONSTANT “*dbbSelectTrunkLayer*”) set to 1 indicates the guide is a select trunk layer.
- Bit 5 (CONSTANT “*dbbTwoTrunksInChannel*”) set to 1 indicates the guide is a two trunks in a channel.
- Bit 6 (CONSTANT “*dbbPinExtendToBoundary*”) set to 1 indicates the guide should extend to the cell boundary.
- Bit 13 (CONSTANT “*dbbStichGuide*”) set to 1 indicates the guide should stitch.
- Bit 14 (CONSTANT “*dbbValidGuide*”) set to 1 indicates the guide is valid.

bbox

See “[bbox](#)” on page A-4.

trunkWidth

The *trunkWidth* field contains the width of the trunk.

trunkLayer

The *trunkLayer* field contains the number of the trunk layer.

downLeftObjectId

The *downLeftObjectId* field contains the object ID of the object to the left of or below the guide.

upRightObjectId

The *upRightObjectId* field contains the object ID of the object to the right of or below the guide.

netId

The *netId* field contains the object ID that is a net object ID. This object ID is part of this net ID's net.

IGRPath Record

An *IGRPath* record contains the fields described in the following sections.

horizontalWidth

The *horizontalWidth* field contains the horizontal width of the interregion global routing path.

verticalWidth

The *verticalWidth* field contains the vertical width of the interregion global routing path.

bbox

See “[bbox](#)” on page [A-4](#) for definition.

classOrNetId

The *classOrNetId* field contains the object ID of the class or net to which this *IGRPath* record is attached.

flags

The *flags* field is internal to Synopsys applications.

horizontalLayer

The *horizontalLayer* field identifies the horizontal layer where this *IGRPath* resides.

numberOfPoint

The *numberOfPoint* field contains an integer comprising the number of points in the *IGRPath*.

points

The points field contains the coordinate value (x,y) of the points in the *IGRPath* record.

verticalLayer

The *verticalLayer* field identifies the vertical layer where this *IGRPath* resides.

PreRouteSegment Record

A *PreRouteSegment* record contains the fields described in the following sections.

dataType

This field is currently designated as internal to Synopsys applications.

statusFlag

The *statusFlag* field is internal to Synopsys applications. When the *statusFlag* field is not 0, one of the following can be specified:

- Bit 0 (CONSTANT “*dbbPreRouteVisible*”) set to 1 indicates the preroute segment is visible.
- Bit 1 (CONSTANT “*dbbCrossRowStrap*”) set to 1 indicates a cross row strap.
- Bit 2 (CONSTANT “*dbbTrunkInTopRightChannel*”) set to 1 indicates the trunk in the top right channel.

- Bit 3 (CONSTANT “*sdbbTrunkInBottomLeftChannel*”) set to 1 indicates the trunk in the bottom left channel.
- Bit 4 (CONSTANT “*dbbPowerTrunkFirst*”) set to 1 indicates the preroute segment should start at the power trunk.
- Bit 5 (CONSTANT “*dbbTrunkBlocked*”) set to 1 indicates the trunk is blocked.
- Bit 6 (CONSTANT “*dbbMeshTrunk*”) set to 1 indicates a mesh trunk.
- Bit 7 (CONSTANT “*dbbPreRouteYDir*”) set to 1 indicates the preroute segment is in the vertical direction.

layer

See “[layer](#)” on page A-9.

offsetOrRegionNumber

One of the following:

- If the preroute segment is either a cross row strap or mesh, the *offsetOrRegionNumber* field contains a cell region number.
- When the preroute segment is outside the region, the *offsetOrRegionNumber* field contains the offset from the channel center.

matchEdgeNumber

The *matchEdgeNumber* field contains the number of the edge record that corresponds to this preroute segment record.

startPoint

The *startPoint* field contains a point coordinate that indicates the lower left starting point of the preroute segment.

endPoint

The *endPoint* field contains a point coordinate that indicates the upper right end point of the preroute segment.

trunkWidth

The *trunkWidth* field indicates the width of the preroute segment in database units.

branchWidthOrCellRowId

The *branchWidthOrCellRowId* field indicates one of the following:

- The width of the pins connected to the preroute segment
- The cell row ID connected to the start of the preroute segment. A cell row ID is specified for maintenance purposes. If the data is compacted, this relationship prevents movement of the preroute segment.

netId

The *netId* field contains the object ID that is a net object ID. This object ID is part of this net ID's net.

newFlag

The *newFlag* field is internal to Synopsys applications. It is a bit field defined as

- Bit 0 (CONSTANT “*dbbHStrapToBoundary*”) set to 1 indicates the horizontal strap should be extended to the boundary.
- Bit 1 (CONSTANT “*dbbVStrapToBoundary*”) set to 1 indicates the vertical strap should be extended to the boundary.
- Bit 2 (CONSTANT “*dbbExtendStrapLowLeftOnly*”) set to 1 restricts the boundary extension to the lower left only.

- Bit 3 (CONSTANT “*dbbExtendStrapUpRightOnly*”) set to 1 restricts the boundary extension to the upper right only.
-

SlicingTreeNode Record

A *SlicingTreeNode* record contains the fields described in the following sections.

cutDirection

One of the following:

- 0 (CONSTANT “*dbcHorizontalCutDir*”) indicates a horizontal cut.
- 1 (CONSTANT “*dbcVerticalCutDir*”) indicates a vertical cut.
- 2 (CONSTANT “*dbcNoSpecifiedCutDir*”) leaves the cut direction unspecified.

nodeNumber

The *nodeNumber* field contains the number of the nodes in the slicing tree.

channelNumber

The *channelNumber* field is internal to Synopsys applications. It contains the number of the channel corresponding to the *SlicingTreeNode* record. 0 indicates that there is no corresponding channel.

parentNodeNumber

The *parentNodeNumber* field contains the number of the *SlicingTreeNode* record’s parent node. 0 indicates the parent node does not exist.

lowLeftChildNumber

The *lowLeftChildNumber* field contains the number of the *SlicingTreeNode* record's lower left child node. 0 indicates that there is not a lower left child node associated with the slicing tree node record.

upRightChildNumber

The *upRightChildNumber* field contains the number of the *SlicingTreeNode* record's upper right child node. 0 indicates that there is not an upper right child node associated with the *SlicingTreeNode* record.

startPoint

The *startPoint* field contains the coordinates of the lower left starting point of the cut line that correspond to the *SlicingTreeNode* record. If the *startPoint* field is not defined, the cut line does not exist.

endPoint

The *endPoint* field contains the coordinates of the upper right end point of the cut line corresponding to the *SlicingTreeNode* record. If the *endPoint* field is not defined the cut line does not exist.

objectId

The *objectId* field contains the database ID of the object that corresponds to the *SlicingTreeNode* record, for example cell region or block, in the database. The object does not exist if the null set is specified.

Placement and Route Records

Boundary Record

A Boundary record contains the field described in the following section.

drawable

The *drawable* field identifies whether a boundary is drawable.

flag

The *flag* record is internal to Synopsys applications. See “[flag](#)” on [page A-8](#) for definition.

dataType

The *dataType* field ...

layer

See “[layer](#)” on [page A-9](#) for definition.

type

The *type* field defines which of the two Boundary types is used. The two types comprise:

- CellBoundary
- PRBoundary

bbox

See “[bbox](#)” on [page A-4](#) for definition.

numberOfPoint

The *numberOfPoint* field contains an integer number of points in the Boundary record.

points

The *points* field contains the coordinate (x,y) of the points in the Boundary record.

CellRegion Record

A CellRegion record contains the fields described in the following sections.

cellInstGroupId

This field is currently designated as internal to Synopsys applications.

cellRowPattern

This field is currently designated as internal to Synopsys applications.

name

The *name* field contains the user specified text string identifying this record.

numberOfCellInst

The *numberOfCellInst* field contains an integer number of CellInst that is linked to this cellRegion record.

cellRowDirection

One of the following:

- Zero (CONSTANT “*dbcHorizontalCellRegion*”) indicates horizontal.
- One (CONSTANT “*dbcVerticalCellRegion*”) indicates vertical.

numberOfCellRow

The *numberOfCellRow* field indicates the number of cell rows in this cell region.

bbox

See “[bbox](#)” on page A-4.

numberOfRegionPin

The *numberOfRegionPin* contains a count of all the region pins located in this region.

numberOfRegionPort

The *numberOfRegionPort* contains a count of all the region ports located in this region.

utilization

The *utilization* contains the projected percentage of standard cell area that this cell region can contain X 100. This field is application maintained.

cellRegionNumber

See “[cellRegionNumber](#)” on page A-7.

CellRow Record

The CellRow record has a run-time hash table that contains information about how the cell instance records can be related to the cell row. This information is temporarily stored in virtual memory. The actual relationship between the row record and its cell instances is established after placement.

A CellRow record contains the fields described in the following sections.

cellRatio

This field is currently designated as internal to Synopsys applications.

index

This field is currently designated as internal to Synopsys applications.

linkCellRowId

This field is currently designated as internal to Synopsys applications.

orientation

The *orientation* field contains a binary number that indicates the orientation of the cell row.

- Bit 0 (CONSTANT “*dbbNoYMirrorCellRow*”) set to 1 indicates a non-mirrored vertical row.
- Bit 1 (CONSTANT “*dbbYMirrorCellRow*”) set to 1 indicates a mirrored vertical row.

- Bit 4 (CONSTANT “*dbbNoXMirrorCellRow*”) set to 1 indicates a non-mirrored horizontal row.
- Bit 5 (CONSTANT “*dbbXMirrorCellRow*”) set to 1 indicates a mirrored horizontal row.

cellClass

The *cellClass* field is an 8-bit field that indicates the cell classes that are allowed in a row. If 0, any cell is allowed. If non-zero, only cells with the same *cellClass* are allowed.

cellAlignment

The *cellAlignment* field contains a number that indicates how cells are aligned in the row:

- 0 (CONSTANT “*dbcCellRowLowLeft*”) aligns the bottom of each cell with the bottom of the cell row.
- 1 (CONSTANT “*dbcCellRowUpLeft*”) aligns the top of each cell with the top of the cell row.
- 2 (CONSTANT “*dbcCellRowOrigin*”) aligns the origin of each cell with the bottom of the cell row.
- 3 (CONSTANT “*dbcCellRowCenter*”) aligns the center of each cell with the center of the cell row.
- 4 (CONSTANT “*dbcCellRowPower*”) aligns the top of the power pin in each cell with the top of the cell row.
- 5 (CONSTANT “*dbcCellRowGnd*”) aligns the bottom of the ground pin in each cell with the bottom of the cell row.
- 6 (CONSTANT “*dbcCellRowPGCenter*”) aligns the point halfway between the bottom of the power pin and the top of the ground pin in each cell with the center of the cell row.

bbox

See “[bbox](#)” on page A-4.

cellRegionId

The *cellRegionId* field contains the object ID of the region where this cell row is located.

linkCellRowId

The *linkCellRowId* field is used in double-back cell row configuration to indicate this cell row’s sibling row.

density

The *density* field is assigned by the set density command.

minDensity

The *minDensity* field is assigned by the set density command.

maxDensity

The *maxDensity* field is assigned by the set density command.

allowablePattern

See “[allowablePattern](#)” on page A-3.

statusFlag

The *statusFlag* field is internal to Synopsys applications.

tile

See “[tile](#)” on page A-12.

GateSite Record

A *GateSite* record contains the following fields:

symmetry

A bit field containing placement information:

- Bit 0 set to 1 places the gate site symmetric against the x axis.
(The gate is mirrored about the y axis.)
- Bit 1 set to 1 places the gate site symmetric against the y axis.
(The gate is mirrored about the x axis.)
- Bit 2 set to 1 rotates the gate site 90 degrees.
- Bit 3 set to 1 rotates the gate site 180 degrees.
- Bit 4 set to 1 rotates the gate site 270 degrees.

layer

See “[layer](#)” on page A-9.

width

A positive integer that defines the width of the site.

height

A positive integer that defines the height of the site.

name

A text string, determined by the user, that references the gate site.

GateSiteArray Record

A *GateSiteArray* record contains the following fields:

justification

This field is unused.

siteTransform

The transformation of the site used in this array.

statusFlag

The statusFlag field is internal to Synopsys applications. It is one of the following:

- 0 indicates that the array is a gate site array.
- 1 indicates that the gate site can be placed on the array.
- 2 indicates that the gate site can not occupy a tile on the array.

bbox

See “[bbox](#)” on page A-4.

gateSiteId

Internal; the object ID of the gate site referenced by this gate site array record.

placementPoint

The origin point of the array.

xDisplacement

The distance between each site in the x direction.

yDisplacement

The distance between each site in the y direction.

xTimes

Number of sites created in the x direction.

yTimes

Number of sites created in the y direction.

RegionBlockage Record

A *RegionBlockage* record contains the fields described in the following sections.

height

The *height* field contains an integer number of the height's value of the *regionBlockage*.

regionId

The *regionId* field contains the object ID of the region that refers to the *regionBlockage*.

width

The *width* field contains an integer number of the width's value of the *regionBlockage*.

xAnchor

This field is currently designated as internal to Synopsys applications.

xOffset

This field is currently designated as internal to Synopsys applications.

yAnchor

This field is currently designated as internal to Synopsys applications.

yOffset

This field is currently designated as internal to Synopsys applications.

RegionPin Record

A *RegionPin* record contains the fields described in the following sections.

externalNetIdx

internalNetIdx

pinConstraintId

statusFlag

The *statusFlag* field is internal to Synopsys applications. See “[Pin/Region Pin statusFlag](#)” on page A-10.

accessDirection

See “[accessDirection](#)” on page A-2.

dataType

The *dataType* field is currently unused.

bbox

See “[bbox](#)” on page A-4.

netId

The *netId* field contains the object ID that is a net object ID. This object ID is part of this net ID’s net.

regionNetId

Along with the *netId*, the *regionNetId* field describes the nets that are associated with the region pin. The region pin/port records bridge the nets of the top level with the nets of the region. The region net will be a child net of the top *netId*.

portId

The *portId* field contains the object ID of the region port of this pin.

cellRegionId

The *cellRegionId* field contains the object ID of the region where this region pin is located.

pinName

The *pinName* field contains a text string of the name of the pin that is associated with this region pin.

rectAttay.layer

See “[layer](#)” on page A-9.

numberOfPoint

If 0, the pin is square (use bbox).

points

One of the following:

- If positive, the pin is the same as other usages of the point field.
- If negative, the pin consists of $-n/2$ rectangles whose bounding boxes are stored in point.

RegionPort Record

A region port may have multiple region pins associated with it. If a region port has multiple pins, all of the region pins are connected inside the region. The region port object interfaces region nets to top level nets.

A *RegionPort* record contains the fields described in the following sections.

direction

See “[direction](#)” on page [A-7](#).

numberOfPin

Number of pins for this port.

netId

The *netId* field contains the object ID that is a net object ID. This object ID is part of this net ID’s net.

regionNetId

Along with the *netId*, the *regionNetId* field describes the nets associated with the region port. The region pin/port records bridge the nets of the top level with the nets of the region. The region net will be a child net of the top-level net.

cellRegionId

The *cellRegionId* field contains the object ID of the region where this region port exists.

name

The *name* field contains a text string of the name of the port that is associated with this region port.

TilePattern Record

The *TilePattern* record contains the fields described in the following sections.

dataValues

This field is currently designated as internal to Synopsys applications.

name

The *name* field contains the user specified text string identifying this record.

numberOfDataValue

This field is currently designated as internal to Synopsys applications.

numberOfTile

This field is currently designated as internal to Synopsys applications.

tilePatternNumber

This field is currently designated as internal to Synopsys applications.

WireDirection Record

The *WireDirection* record contains the fields described in the following sections.

preferedDirection

One of the following:

- 0 for horizontal tracks
- 1 for vertical tracks
- 2 for not specified

layer

See “[layer](#)” on page A-9.

baseArrayClass

The ID number of the base array class the tracks belong.

WireTrack Record

The *WireTrack* record contains the fields described in the following sections.

trackDirection

One of the following:

- 0 for horizontal tracks
- 1 for vertical tracks

layer

See “[layer](#)” on page A-9.

bbox

See “[bbox](#)” on page A-4.

startTrack

Starting point of the tracks.

trackStep

Distance between each track.

times

Number of tracks created.

type

One of the following:

- 0 for wire track
- 1 for via track

baseArrayClass

The ID number of the base array class the tracks belong.

Polygon Layout Records

BPVContactArray Record

The BPVContactArray record is used for defining user-specified macro feedthroughs and user-specified pin solutions in standard cells. The size of the pseudo contact array is determined by the product of the *xDuplicate* and *yDuplicate* fields.

A BPVContactArray record contains the fields described in the following sections.

netId

The *netId* field contains the object ID for the net to which this *BPVContactArray* record is attached.

position

The *position* field contains the origin position coordinates for the BPVContactArray.

routeType

See “[routeType](#)” on page A-11 for definition.

contactCodeIdx

The *contactCodeIdx* field contains a number that indexes into the contact table in the technology file. Based on this index, the Synopsys application chooses a specific type of contact.

transformCode

See “[transformCode](#)” on page A-13.

flag

The *flag* field is internal to Synopsys applications.

bbox

See “[bbox](#)” on page A-4.

status

The *status* field is a bit field interpreted as an integer. Each bit in the field indicates some aspect of the record. This field is maintained by the application.

accessDirection

See “[accessDirection](#)” on page A-2.

xPitch

The *xPitch* field contains the center-to-center spacing of the contact cuts in the horizontal direction, assuming no transformation.

yPitch

The *yPitch* field contains the center-to-center spacing of the contact cuts in the vertical direction, assuming no transformation.

groupId

The *groupId* field of the group of which this contact is a part.

xDuplicate

The *xDuplicate* field contains the number of contact cuts in the horizontal direction.

yDuplicate

The *yDuplicate* field contains the number of contact cuts in the vertical direction.

bitmap

The flag field indicates whether or not a bitmap field exists. If the bitmap field exists, its size is equal to xTimes * yTimes.

BPVPath Record

The BPVPath record is only used for defining user-specified macro feedthroughs and user-specified pin solutions in standard cells.

A pseudo path record contains the fields described in the following sections.

netId

The *netId* field contains the object ID for the net to which this BPVPath record is attached.

dataType

The *dataType* field is currently unused.

pathType

See “[pathType](#)” on page A-10.

routeType

See “[routeType](#)” on page A-11.

bbox

See “[bbox](#)” on page A-4.

width

The *width* field contains the width of the path in database units.

groupId

The *groupId* field specifies the group of objects of which this pseudo path record is associated.

status

The *status* field is a bit field interpreted as an integer. Each bit in the field indicates some aspect of the record. This field is maintained by the application.

layer

See “[layer](#)” on page A-9.

numberOfPoint

The *numberOfPoint* field contains the number of points in the *points* field.

points

The *points* field contains the points associated with an object.

Contact Record

A Contact record contains the fields described in the following sections.

contactCodeIdx

The *contactCodeIdx* field contains a number that indexes into the contact table in the technology file. The Synopsys application chooses a specific type of contact based on this index.

transformCode

See “[transformCode](#)” on page A-13.

routeType

See “[routeType](#)” on page A-11.

netId

The *netId* field contains the object ID that is a net object ID. This object ID is part of this net ID’s net.

centerPosition

The *centerPosition* field contains the point at which the center of the contact is placed. The Scheme interface reports the center in user units. The C interface reports the center in database units.

ContactArray Record

The size of the contact array is determined by the product of the *xtimes* and *ytimes* fields.

A ContactArray record contains the fields described in the following sections.

routeType

See “[routeType](#)” on page A-11 for definition.

contactCodidx

The *contactCodidx* field contains a number that indexes into the contact table in the technology file. The Synopsys application chooses a specific type of contact based on this index.

transformCode

See “[transformCode](#)” on page A-13.

flag

The flag field is internal to Synopsys applications.

Bit 0 indicates the status of the bitmap. It can be one of the following:

- If set to 0, no bitmap is present.
- If set to 1, bitmap is present.

bbox

See “[bbox](#)” on page A-4.

position

The *position* field contains the point at which the center of the contact array is placed. The Scheme interface reports the *centerPoint* in user units. The C interface reports the *centerPoint* in database units.

xPitch

The *xPitch* field contains the center-to-center spacing of the contact cuts in the horizontal direction, assuming no transformation.

yPitch

The *yPitch* field contains the center-to-center spacing of the contact cuts in the vertical direction, assuming no transformation.

netId

The *netId* field contains the object ID that is a net object ID. This object ID is part of this net ID’s net.

xDuplicate

The *xDuplicate* field contains the number of contact cuts in the horizontal direction.

yDuplicate

The *yDuplicate* field contains the number of contact cuts in the vertical direction.

bitmap

The *bitmap* field indicates whether or not a bitmap field exists. If the bitmap field exists, its size is equal to *xTimes* * *yTimes* bits.

ExpandedPolygon Record

An *ExpandedPolygon* record contains the fields described in the following sections.

bbox

See “[bbox](#)” on page [A-4](#) for definition.

dataType

This field is currently designated as internal to Synopsys applications.

layer

See “[layer](#)” on page [A-9](#) for definition.

netId

The *netId* field contains the object ID for the net to which this *ExpandedPolygon* record is attached.

numberOfPoint

The *numberOfPoint* field contains the number of points that create a polygon within the *ExpandedPolygon* record.

points

The *points* field contains coordinates for the points that define a polygon within an *ExpandedPolygon* record.

HorizontalWire Record

Wires provide a compact method of representing routing data. To discover the width and layer of a wire, use “[dbMakeWireMasterId](#)” on [page 4-190](#) to create the wire’s master object ID, then fetch the master.

A *HorizontalWire* record contains the fields described in the following sections. A *HorizontalWire* record is always for a horizontal wire.

layer

See “[layer](#)” on [page A-9](#) for definition.

regionNumber

The *regionNumber* field contains the number of the region in which this wire exists. A zero (0) implies an inter-region wire.

routeType

See “[routeType](#)” on [page A-11](#).

wireMasterIdx

The *wireMasterIdx* field contains the object ID of the wire’s descriptor record.

length

The value contained in the *length* field represents the length of the wire in the x direction. This value is specified in database units and can be either positive or negative. A positive value indicates that the wire goes to the right from the coordinate specified in the *startPoint* field and a negative value indicates that the wire goes to the left.

netId

The *netId* field contains the object ID that is a net object ID. This object ID is part of this net ID's net.

startPoint

The *startPoint* field contains an (x y) coordinate associated with the starting point of the wire.

Path Record

A Path record contains the fields described in the following sections.

dataType

The *dataType* field is currently unused.

pathType

See “[pathType](#)” on page A-10.

routeType

See “[routeType](#)” on page A-11.

bbox

See “[bbox](#)” on page A-4.

width

The *width* field contains the width of the path in database units.

netId

The *netId* field contains the object ID that is a net object ID. This object ID is part of this net ID's net.

layer

See “[layer](#)” on page A-9.

numberOfPoint

The *numberOfPoint* field contains the number of points in the *point* field.

points

The points comprising the path.

Polygon Record

The Synopsys application uses the Polygon record to represent the input GDSII Stream data prior to pin and blockage extraction. This record has limited use during placement and routing because of the lack of net information.

A polygon record contains the fields described in the following sections.

netId

The *netId* field contains the object ID for the net to which this Polygon record is attached.

dataType

The *dataType* field is currently unused.

layer

See “[layer](#)” on page A-9.

bbox

See “[bbox](#)” on page A-4.

numberOfPoint

The *numberOfPoint* field contains the number of unpacked points within this Polygon record.

points

The *points* field contains the points comprising the polygon.

Rectangle Record

The Rectangle record represents the input GDSII Stream data prior to pin and blockage extraction. This record has limited use during placement and routing because of the lack of net information.

A rectangle record contains the fields described in the following sections.

netId

The *netId* field contains the object ID for the net to which this Rectangle record is attached.

dataType

The *dataType* field is currently unused.

layer

See “[layer](#)” on page A-9.

bbox

See “[bbox](#)” on page A-4.

RectArray Record

A *RectArray* record contains the fields described in the following sections.

bbox

See “[bbox](#)” on page A-4 for definition

dataType

This field is currently designated as internal to Synopsys applications.

layer

See “[layer](#)” on page A-9 for definition.

numberOfPoint

The *numberOfPoint* field contains a negative number which means that ((-*numberOfPoint*)/2) rectangle in the *RectArray*.

points

The *points* field contains the coordinate value (x,y) of the points in the *rectArray* record.

Text Record

A Text record contains the fields described in the following sections.

dataType

The *dataType* field is currently unused.

fontId

The *font_id* field contains a number from 0 to 255 that indicates the font to be used for text.

tjust

The *tjust* field Indicates the justification of the text as follows:

- 0 (CONSTANT “*dbcLowerLeft*”) places the text to the lower left.
- 1 (CONSTANT “*dbcLowerCenter*”) places the text to the lower center.
- 2 (CONSTANT “*dbcLowerRight*”) places the text to the lower right.
- 3 (CONSTANT “*dbcCenterLeft*”) places the text to the center left.
- 4 (CONSTANT “*dbcCenterCenter*”) places the text to the center center.
- 5 (CONSTANT “*dbcCenterRight*”) places the text to the center right.
- 6 (CONSTANT “*dbcUpperLeft*”) places the text to the upper left.
- 7 (CONSTANT “*dbcUpperCenter*”) places the text to the upper center.

- 8 (CONSTANT “*dbcUpperRight*”) places the text to the upper right.

bbox

See “[bbox](#)” on page A-4.

position

The *position* field indicates the x,y coordinate of the text.

text_height

The *text_height* field indicates the height of the text in database units.

layer

See “[layer](#)” on page A-9.

transformCode

See “[transformCode](#)” on page A-13.

stringId

The *stringId* field contains a text string that indicates the value of the record.

VerticalWire Record

Wires provide a compact method of representing routing data. To discover the width and layer of a wire, use “[dbMakeWireMasterId](#)” on page 4-190 to create the wire’s master object ID, then fetch the master. The master will describe the width and pathtype information.

An internal Y wire record contains the fields described in the following sections. A Y wire object is always a vertical wire.

regionNumber

The *regionNumber* field contains the number of the region in which this wire exists. A zero (0) implies an inter-region wire.

routeType

See “[routeType](#)” on page [A-11](#).

wireMasterIdx

The *internalWireMasterId* field contains the object ID of the wire’s descriptor record.

length

The value contained in the *length* field represents the length of the wire in the y direction. This value is specified in database units and can be either positive or negative. A positive value indicates that the wire goes up from the coordinate specified in the *startPoint* field and a negative value indicates that the wire goes down.

netId

The *netId* field contains the object ID that is a net object ID. This object ID is part of this net ID’s net.

startPoint

The *startPoint* field contains an (x y) coordinate associated with the starting point of the wire.

layer

See “[layer](#)” on page [A-9](#) for definition.

ViaRegion Record

A *ViaRegion* record contains the fields described in the following sections.

contactCodeIdx

This field is currently designated as internal to Synopsys applications.

pinId

The *pinId* field contains the object ID of the Pin object linked to the *ViaRegion* record.

portId

The *portId* field contains the object ID of the Port object linked to the *ViaRegion* record.

bbox

See “[bbox](#)” on page [A-4](#) for definition

dataType

This field is currently designated as internal to Synopsys applications.

layer

See “[layer](#)” on page [A-9](#) for definition.

numberOfPoint

The *numberOfPoint* field contains the number of points that create a rectangle within the *rectArray* of a *viaRegion* record.

points

The *points* field indicates the x and y coordinates of the points of the *rectArray* of the *viaRegion* record.

rotate90

This field is currently designated as internal to Synopsys applications.

userMarked

This field is currently designated as internal to Synopsys applications.

WireMaster Record

A *WireMaster* record contains the fields described in the following sections.

index

This field is currently designated as internal to Synopsys applications.

dataType

The *dataType* field is currently unused in Milkyway.

The *dataType* field identifies the status of the vias in the sea of gates.

- If bit 0 is set, the wire via is a power/ground ring.
- If bit 1 is set, the wire via is a power/ground stripe.
- If bit 2 is set, the wire via is placed during power/ground routing.

- If bit 3 is set, the placement of the wire via is determined by the default values.
- If bit 4 is set, the placement of the wire via is fixed.

pathType

See “[pathType](#)” on page A-10.

status

The *status* field is a bit field interpreted as an integer. This field is maintained by the application. If bit 0 is set to 1 the wire is prewired.

width

The *width* field indicates width of the wire in database units.

layer

See “[layer](#)” on page A-9.

Timing Related Records

3ValueSet Record

A *3ValueSet* record contains the fields described in the following sections.

max

The max field indicates the maximum value

min

The min field indicates the minimum value

nom

The nom field indicates the normal value

4ValueSet

A *4ValueSet* record contains the fields described in the following sections.

fallMax

The fallMax field indicates the maximum value at the fall edge.

fallMin

The fallMin field indicates the minimum value at the fall edge.

riseMax

The riseMax field indicates the maximum value at the rise edge.

riseMin

The riseMin field indicates the minimum value at the rise edge.

6ValueSet

fallMax

The *fallMax* field indicates the maximum value at the fall edge.

fallMin

The *fallMin* field indicates the minimum value at the fall edge.

fallNom

The *fallNom* field indicates the normal value at the fall edge.

riseMax

The *riseMax* field indicates the maximum value at the rise edge.

riseMin

The *riseMin* field indicates the minimum value at the rise edge.

riseNom

The *riseNom* field indicates the normal value at the rise edge.

ClkOptData Record

A *ClkOptData* record contains the fields described in the following sections.

invertDel

This field is currently designated as internal to Synopsys applications.

level

This field is currently designated as internal to Synopsys applications.

nonInvDel

This field is currently designated as internal to Synopsys applications.

phaseFlag

The *phaseFlag* field is internal to Synopsys applications.

portName

The *portName* field contains a text string specifying the port that this *ClkOptData* record references.

signature

This field is currently designated as internal to Synopsys applications.

statusFlag

The *statusFlag* field is internal to Synopsys applications.

DelayTable**index1Type**

This field is currently designated as internal to Synopsys applications.

index2Type

This field is currently designated as internal to Synopsys applications.

index3Type

This field is currently designated as internal to Synopsys applications.

lengthIndex1

This field is currently designated as internal to Synopsys applications.

lengthIndex2

This field is currently designated as internal to Synopsys applications.

lengthIndex3

This field is currently designated as internal to Synopsys applications.

name

The *name* field contains the user specified text string identifying this record.

tableType

This field is currently designated as internal to Synopsys applications.

valueOrValueTable

This field is currently designated as internal to Synopsys applications.

PortFunc Record

A *PortFunc* record contains the fields described in the following sections.

condStrId

This field is currently designated as internal to Synopsys applications.

type

This field is currently designated as internal to Synopsys applications.

PortToPortDelay Record

A *PortToPortDelay* record contains the fields described in the following sections.

endCondid

This field is currently designated as internal to Synopsys applications.

recType

This field is currently designated as internal to Synopsys applications.

startCondid

This field is currently designated as internal to Synopsys applications.

unateness

One of the following:

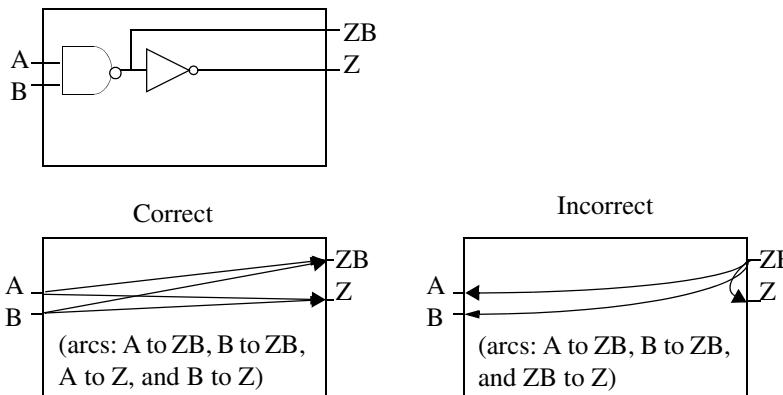
- 0 (CONSTANT “*dbcUnateNonInvert*”) indicates positive unateness. For example, a rising edge on the *fromPortId* will cause a rising edge on the *toPortId*.
- 1 (CONSTANT “*dbcUnateInvert*”) indicates negative unateness. For example, a rising edge on the *fromPortId* will cause a falling edge on the *toPortId*.

- 2 (CONSTANT “*dbcNonUnate*”) indicates non-unateness. The *toPortId* edge direction cannot be determined by the *fromPortId* edge direction.

timingType

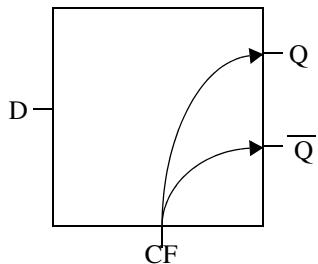
This field contains the timing type of the arc. Timing type can be either intrinsic delay or timing rule.

- If the *timingType* field is set to 0, path, an intrinsic delay type, is specified.
 - *Path* (standard combinatorial delay) indicates the arc type shown below.



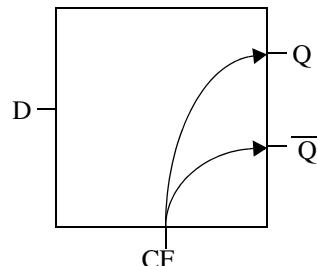
- If the *timingType* field is set to 1, *clock_rising*, an intrinsic delay type, is specified.
 - *Clock_rising* indicates the arc type shown below.

Positive-edge FlipFlop



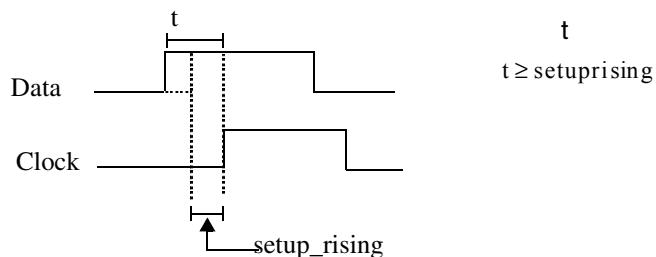
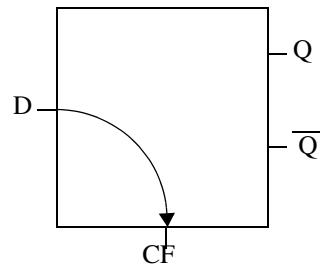
- If the *timingType* field is set to 2, *clock_falling*, an intrinsic delay type, is specified.
 - *Clock_falling* indicates the arc type shown below.

Negative-edge FlipFlop



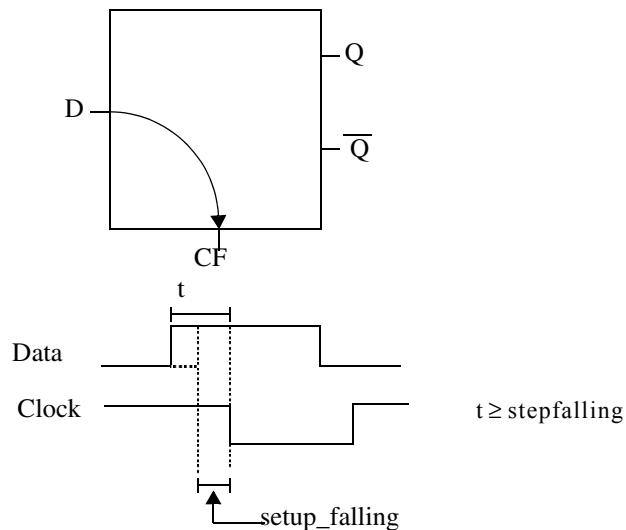
- If the *timingType* field is set to 3, *setup_rising*, a timing rule type, is specified.
 - *Setup_rising* indicates the arc type shown below.

Positive-edge Triggered FlipFlop



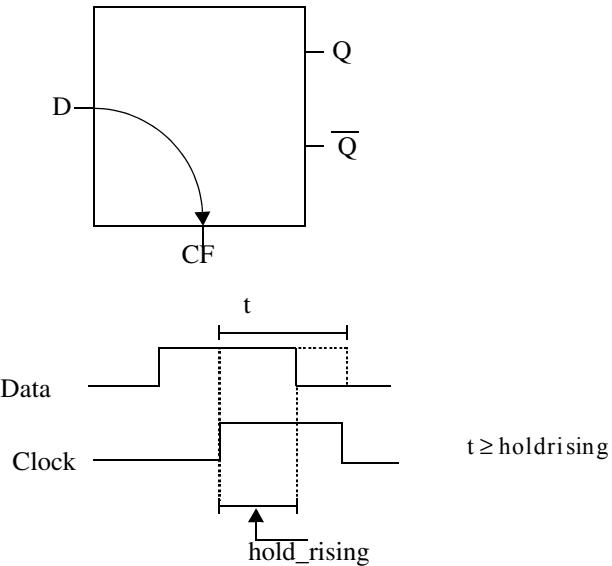
- If the *timingType* field is set to 4, *setup_falling*, a timing rule type, is specified.
 - *Setup_falling* indicates the arc type shown below.

Negative-edge Triggered FlipFlop



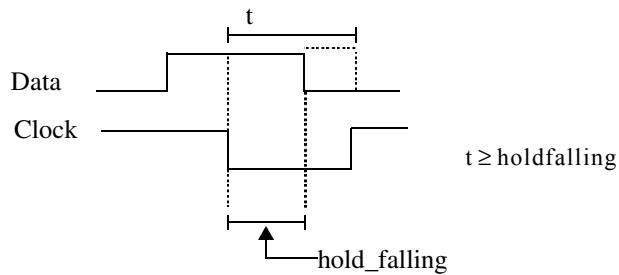
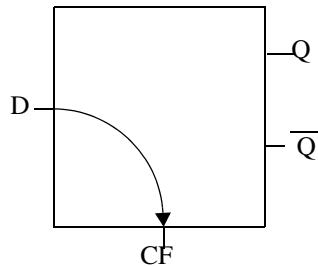
- If the *timingType* field is set to 5, *hold_rising*, a timing rule type, is specified.
 - *Hold_rising* indicates the arc type shown below.

Positive-edge Triggered FlipFlop



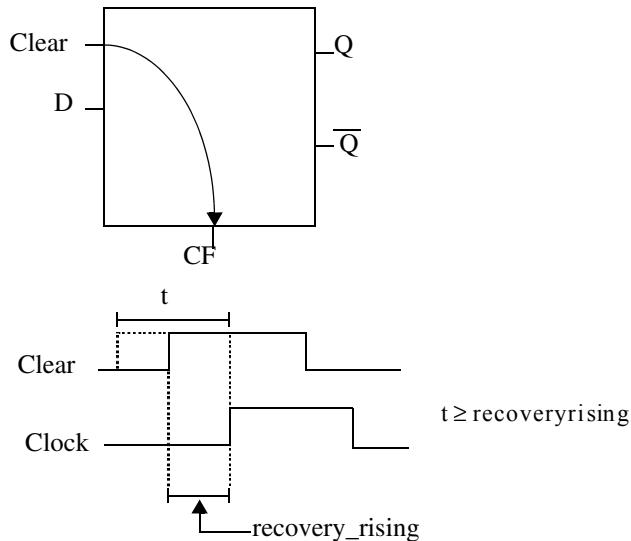
- If the *timingType* field is set to 6, *hold_falling*, a timing rule type, is specified.
 - *Hold_falling* indicates the arc type shown below.

Negative-edge Triggered FlipFlop



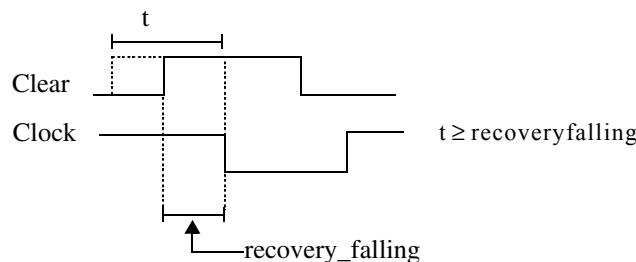
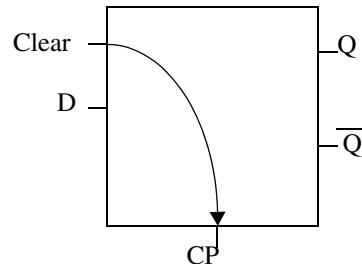
- If the *timingType* field is set to 7, *recovery_rising*, a timing rule type, is specified.
 - *Recovery_rising* indicates the arc type shown below.

Positive-edge Triggered FlipFlop

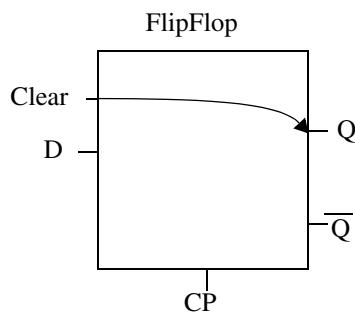


- If the *timingType* field is set to 8, *recovery_falling*, a timing rule type, is specified.
 - *Recovery_falling* indicates the arc type shown below.

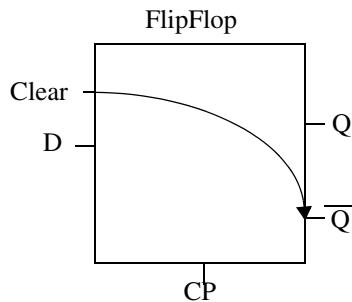
Negative-edge Triggered FlipFlop



- If the *timingType* field is set to 9, clear, an intrinsic delay type, is specified.
 - *Clear* indicates the arc type shown below.



- If the *timingType* field is set to 10, preset, an intrinsic delay type, is specified.
 - *Preset* indicates the arc type shown below.



fromPortId

The *fromPortId* field contains the source *portObjectId* of this delay arc.

toPortId

The *toPortId* field contains the destination *portObjectId* of this delay arc.

fallIntrinsicMin

The *fallIntrinsicMin* field contains the minimum delay time from *fromPortId* to *toPortId* when output signal is falling.

fallIntrinsicMax

The *fallIntrinsicMax* field contains the maximum delay time from *fromPortId* to *toPortId* when output signal is falling.

riseIntrinsicMin

The *riseIntrinsicMin* field contains the minimum delay time from *fromPortId* to *toPortId* when output signal is rising.

riseIntrinsicMax

The *riseIntrinsicMax* field contains the maximum delay time from *fromPortId* to *toPortId* when output signal is rising.

fallResistMin

The *fallResistMin* field contains the minimum output resistance from *fromPortId* to *toPortId* when output signal is falling.

fallResistMax

The *fallResistMax* field contains the maximum output resistance from *fromPortId* to *toPortId* when output signal is falling.

riseResistMin

The *riseResistMin* field contains the minimum output resistance from *fromPortId* to *toPortId* when output signal is rising.

riseResistMax

The *riseResistMax* field contains the maximum output resistance from *fromPortId* to *toPortId* when output signal is rising.

fallSlopeMin

The *fallSlopeMin* field contains the minimum slope delay when output signal is falling.

fallSlopeMax

The *fallSlopeMax* field contains the maximum slope delay when output signal is falling.

riseSlopeMin

The *riseSlopeMin* field contains the minimum slope delay when output signal is rising.

riseSlopeMax

The *riseSlopeMax* field contains the maximum slope delay when output signal is rising.

fallIntrinsicNom

The *fallIntrinsicNom* field contains the nominal delay time from *fromPortId* to *toPortId* when output signal is falling.

riseIntrinsicNom

The *riseIntrinsicNom* field contains the nominal delay time from *fromPortId* to *toPortId* when output signal is rising.

fallResistNom

The *fallResistNom* field contains the nominal output resistance from *fromPortId* to *toPortId* when output signal is falling.

riseResistNom

The *riseResistNom* field contains the nominal output resistance from *fromPortId* to *toPortId* when output signal is rising.

fallSlopeNom

The *fallSlopeNom* field contains the nominal slope delay when output signal is falling.

riseSlopeNom

The *riseSlopeNom* field contains the nominal slope delay when output signal is rising.

PortToPortTable Record

A *PortToPortTable* record contains the fields described in the following sections.

endCondId

This field is currently designated as internal to Synopsys applications.

fromPortId

This field is currently designated as internal to Synopsys applications.

maxFallPropDelayId

This field is currently designated as internal to Synopsys applications.

maxFallTranDelayId

This field is currently designated as internal to Synopsys applications.

maxRisePropDelayId

This field is currently designated as internal to Synopsys applications.

maxRiseTranDelayId

This field is currently designated as internal to Synopsys applications.

minFallPropDelayId

This field is currently designated as internal to Synopsys applications.

minFallTranDelayId

This field is currently designated as internal to Synopsys applications.

minRisePropDelayId

This field is currently designated as internal to Synopsys applications.

minRiseTranDelayId

This field is currently designated as internal to Synopsys applications.

nomFallPropDelayId

This field is currently designated as internal to Synopsys applications.

nomFallTranDelayId

This field is currently designated as internal to Synopsys applications.

nomRisePropDelayId

This field is currently designated as internal to Synopsys applications.

nomRiseTranDelayId

This field is currently designated as internal to Synopsys applications.

recType

This field is currently designated as internal to Synopsys applications.

startCondId

This field is currently designated as internal to Synopsys applications.

status

This field is currently designated as internal to Synopsys applications.

timingType

This field is currently designated as internal to Synopsys applications.

toPortId

This field is currently designated as internal to Synopsys applications.

unateness

This field is currently designated as internal to Synopsys applications.

SlackGraphEdge Record

The *SlackGraphEdge* record is used to record the slack graph for the design. Each edge is a directed graph edge containing signal flow information. The record's fields are described below.

status

The status field is currently unused.

fromObjectId

The *fromObjectId* field contains the source *objectId* for this timing arc.

toObjectId

The *toObjectId* field contains the destination *objectId* for this timing arc.

SlackGraphPrimaryNode Record

The *SlackGraphPrimaryNode* records define the start and ending points for the directed graph. Edge record will emanate from and terminate on these nodes. The record's fields are described below.

status

One of the following:

- 0 indicates that this node record specifies the node arrival time.
- 1 (CONSTANT “*dbbOutputPrimaryNode*”) indicates that this node record specifies the node required arrival time.

startOrEndObjectId

The *startOrEndObjectId* field contains the object ID of either the starting point or the end point associated with the record. These objects will be either a port instance or a port.

riseMax

One of the following:

- If the status field is 0, then this field represents the latest time the signal will arrive on a rising edge.
- If the status field is 1, then this field represents the latest time the signal should arrive on a rising edge.

fallMax

One of the following:

- If the status field is 0, then this field represents the latest time the signal will arrive on a falling edge.
- If the status field is 1, then this field represents the latest time the signal should arrive on a falling edge.

riseMin

One of the following:

- If the status field is 0, then this field represents the earliest time the signal will arrive on a rising edge.
- If the status field is 1, then this field represents the earliest time the signal should arrive on a rising edge.

fallMin

One of the following:

- If the status field is 0, then this field represents the earliest time the signal will arrive on a falling edge.
- If the status field is 1, then this field represents the earliest time the signal should arrive on a falling edge.

Other Records

BaseArray Record

A BaseArray record contains the fields described in the following sections.

layer

See “[layer](#)” on page [A-9](#) for definition.

orientation

The *orientation* field currently is unused. It is reserved for future use.

baseArrayClass

Internal; the class number assigned to the base array by the technology file.

bbox

The *bbox* field is a common field. For more information see “[bbox](#)” on page [A-4](#).

name

Name of the class assigned to the base array record.

tileWidth

The *tileWidth* field contains the width of a tile in the class. For column-structured arrays, width is the vertical dimension; for row-structured arrays, width is the horizontal dimension.

This field is internal to the Synopsys application. The Synopsys application assigns the width of the tile from the *tileClassTable* section of the technology file. For more information, see “Tile Section” in the *Milkyway Data Preparation User Guide*.

tileHeight

The *tileHeight* field contains the height of a tile in the class. For column-structured arrays, height is the horizontal dimension; for row-structured arrays, height is the vertical dimension.

This field is internal to the Synopsys application. The Synopsys application assigns the height of the tile from the *tileClassTable* section of the technology file. For more information, see “Tile Section” in the *Milkyway Data Preparation User Guide*.

numberOfCellRow

Number of base array rows in this record.

subTileFlag

The *subTileFlag* field is internal to Synopsys applications. It is assigned by the technology file.

CellRow Record

A CellRow record contains the fields described in the following sections.

orientation

One of the following:

- Zero (0) indicates horizontal.
- A non-zero value indicates vertical.

cellClass

The *cellClass* field currently is unused. It is reserved for future use.

bbox

The *bbox* field is a common field. For more information see “[bbox](#)” on page A-4.

baseArrayId

The *baseArrayId* field contains the Database ID of the base array in which this row exists.

densityHundredthOfPercent

Assigned by the set density command.

minDensityHundredthOfPercent

Assigned by the set density command.

maxDensityHundredthOfPercent

Assigned by the set density command.

allowablePattern

The *allowablePattern* field is a common field. For more information see “[allowablePattern](#)” on page A-3.

tile

The *tile* field is a common field. For more information see “[tile](#)” on page A-12.

CellEquivalentClass Record

A CellEquivalentClass record contains the fields described in the following sections.

classType

If 1 (CONSTANT “*dbcCellEEQClass*”), indicates cellName1 and cellName2 are electrically equivalent. If 2 (CONSTANT “*dbcCellLEQClass*”), indicates cellName1 and cellName2 are logically equivalent.

cellName1

Cell name specified is equivalent to cell name specified in cellName2 field.

cellName2

Cell name specified is equivalent to cell name specified in cellName1 field.

InstPower Record

An *InstPower record* contains the fields described in the following sections.

instId

This field is currently designated as internal to Synopsys applications.

power

This field is currently designated as internal to Synopsys applications.

PVTDelayScale

This field is currently designated as internal to Synopsys applications.

temperature

This field is currently designated as internal to Synopsys applications.

NetPower Record

A *NetPower* record contains the fields described in the following sections.

netId

The *netId* field contains the object ID for the net to which this *NetPower* record is attached.

switchActivity

This field is currently designated as internal to Synopsys applications.

temperature

This field is currently designated as internal to Synopsys applications.

PinAccEdge Record

A *PinAccEdge* record contains the fields described in the following sections.

accessDir

See “[accessDirection](#)” on page A-2 for definition

bbox

See “[bbox](#)” on page A-4 for definition

center

This field is currently designated as internal to Synopsys applications.

layer

See “[layer](#)” on page A-9 for definition.

pinId

The *pinId* field contains the object ID of the pin object that refers to the *PinAccEdge* record

width

This field is currently designated as internal to Synopsys applications.

Top Design Format Records

TdfClk Record

A *TdfClk* record contains the fields described in the following sections.

fallMax

The *fallMax* field contains the description of the clock waveform at the falling edge.

fallMin

The *fallMin* field contains the description of the clock waveform at the falling edge.

name

The *name* field contains the user specified text string identifying this record.

period

The *period* field contains the description of the clock waveform.

riseMax

The *riseMax* field contains the description of the clock waveform at the rising edge.

riseMin

The *riseMin* field contains the description of the clock waveform at the rising edge.

skew

The *skew* field contains the value of the clock pin delay.

TdfFlipChipPad Record

A *TdfFlipChipPad* record contains the fields described in the following sections.

name

The *name* field contains a user specified text string identifying this record.

cellInstId

The *cellInstId* field contains the object ID for the chip pad's corresponding pad cellInst.

idxRow

The *idxRow* field contains a chip pad's row index within the chip matrix.

weightRow

The *weightRow* field contains a chip pad's row ordering within the chip matrix.

yoffset

The *yoffset* field contains a chip pad's y offset corresponding to the chip matrix's low left corner.

idxCol

The *idxCol* field contains a chip pad's column index within the chip matrix.

weightCol

The *weightCol* field contains a chip pad's column ordering within the chip matrix.

xoffset

The *xoffset* field contains a chip pad's x offset corresponding to the chip matrix's low left corner.

```
unsigned char pattern; /* chip pad's orientation: R0, R0_MX, R0_MY,  
R180, R90,  
R90_MX, R90_MY, R270 */  
  
unsigned char reserved[24]; /* reserved for future use */  
  
} axTdfFlipChipPad_t;
```

TdfFlipChipMatrix Record

A *TdfFlipChipMatrix* record contains the fields described in the following sections.

name

The *name* field contains the user specified text string identifying this record.

xpitch

The *xpitch* field contains the x direction distance expressed with a real number in user defined units.

ypitch

The *ypitch* field contains the y direction distance expressed with a real number in user defined units.

TdfInputOutput Record

A *TdfInputOutput* record contains the fields described in the following sections.

dataType

The *dataType* field comprises two *datatypes*:

- axcInputdelay
- axcOutputDelay

fallingActive

The *fallingactive* field comprises two bits: *tdfcLatchFlipflopFlag* and *tdffiseFallFlag* and includes four combinations:

- Output Delay
 - rise_flipflop0,0
 - rise_latch1,0
- Input Delay
 - fall_flipflop0,1
 - fall_latch1,1

fallMax

The *fallMax* field contains the delay constraint for maximum fall.

fallMin

The *fallMin* field contains the delay constraint for minimum fall.

name

The *name* field contains the user specified text string identifying this record.

riseMax

The *riseMax* field contains the delay constraint for maximum rise.

riseMin

The *riseMin* field contains the delay constraint for minimum rise.

TdfNet Record

A *TdfNet* record contains the fields described in the following sections.

estimCapacitMax

The *estimCapacitMax* field contains the maximum estimated capacitance constraint.

estimCapacitMin

The *estimCapacitMin* field contains the minimum estimated capacitance constraint.

fallMax

The *fallMax* field contains the delay constraint for maximum fall.

fallMin

The *fallMin* field contains the delay constraint for minimum fall.

fallSetUpMax

The *fallSetUpMax* field contains the maximum transition constraint for the fall setup.

fallSetUpMin

The *fallSetUpMin* field contains the minimum transition constraint for the fall setup.

horWeight

The *horWeight* field contains the horizontal weight, where 1 is no weight. One is the default.

name

The *name* field contains the user defined text string identifying this record.

riseMax

The *riseMax* field contains the delay constraint for maximum rise.

riseMin

The *riseMin* field contains the delay constraint for minimum rise.

riseSetUpMax

The *riseSetUpMax* field contains the maximum transition constraint for the rise setup.

riseSetUpMin

The *riseSetUpMin* field contains the minimum transition constraint for the rise setup.

verWeight

The *verWeight* field contains the vertical weight, where 1 is no weight. One is the default.

TdfPad Record

A *TdfPad* record contains the fields described in the following sections.

depth

The *depth* field indicates the depth of pad intrusion into the cell from the cell edge.

name

The *name* field contains the user defined text string identifying this record.

offset

The *offset* field indicates the offset from bottom-left edge. The value is 0 whenever there is no location constraint.

padOrder

The *padOrder* field starts with 1 (0 whenever there is no order constraint).

padSide

The *padSide* field contains left, right, bottom, and top.

pinLayer

The *pinLayer* field is 255 whenever this is a pad.

status

The *status* field contains the *TdfPad* record, and can include any of the following:

- fixed
- placed
- unplaced
- unfixed

width

The *width* field contains the width of the pad where it abuts the edge.

TdfResCap Record

A *TdfResCap* record contains the fields described in the following sections.

datatype

The *datatype* field comprises three *dataTypes*:

- axcLoadCapacitance
- axcSrcResistance
- axcSrcCapacitance

fallMax

The *fallMax* field indicates the delay constraint maximum fall.

fallMin

The *fallMin* field indicates the delay constraint minimum fall.

fallNom

The *fallNom* field indicates the delay constraint nominal fall.

name

The *name* field contains the user specified text string identifying this record.

riseMax

The *riseMax* field indicates the delay constraint maximum rise.

riseMin

The *riseMin* field indicates the delay constraint minimum rise.

riseNom

The *riseNom* field indicates the delay constraint nominal rise.

TdfTimeRequire Record

A *TdfTimeRequire* record contains the fields described in the following sections.

cmdType

The *cmdType* field contains 0-7bits defined as follows:.

- Bit0false path
- Bit1multicycle path
- Bit2max delay path
- Bit3min delay path
- Bit4time borrow
- Bit5disable timing
- Bit6restore timing
- Bit7reset path

name

The *name* field contains the user specified text string identifying this record.

value

The *value* field contains the value for the timing requirement.

statusFlag

The *statusFlag* field is internal to Synopsys applications. It contains 0-7 bits defined as follows:

- Bit0setup
- Bit1hold
- Bit2setuphold
- Bit3rise
- Bit4fall
- Bit5risefall
- Bit6start
- Bit7end

Cell Library Format Records

CIfPadDef Record

A *CIfPadDef* record contains the fields described in the following sections.

name

The *name* field contains the user specified text string identifying this record.

transform

See “[transformCode](#)” on page A-13 for definition.

CIfRecord Record

A *CIfRecord* record contains the fields described in the following sections.

data

This field is currently designated as internal to Synopsys applications.

dataLength

This field is currently designated as internal to Synopsys applications.

operation

This field is currently designated as internal to Synopsys applications.

DataPath Records

DpsCell Record

A *DpsCell* Record record contains the fields described in the following sections.

assigned

The *assigned* field indicates the flag determining whether a cell is already assigned to a DPCM.

instId

The *instId* field indicates the object ID for the cellInst.

spaceLowLeft

The *spaceLowLeft* field indicates the distance between a cell's left/low edge and a slot's left/low ledge.

spaceUpRight

The *spaceUpRight* field indicates the distance between a cell's up/right edge and a slots up/right edge.

DpsDpcm Record

A *DpsDpcm* record contains the fields described in the following sections.

bbox

See “[bbox](#)” on page [A-4](#) for definition.

columns

The *columns* field contains the number of columns.

name

The *name* field contains the user specified text string identifying this record.

rows

The *rows* field contains the number of rows.

DpsSlot Record

A *DpsSlot* record contains the fields described in the following sections.

collIndex

The *collIndex* field contains the number of columns within the DPCM.

rowStack

The *rowIndex* field contains the number of rows within the DPCM.

bbox

See “[bbox](#)” on page [A-4](#) for definition.

name

The *name* field contains the user specified text string identifying this record.

rowStack

The *rowStack* field indicates the multiple height of a row.

Plan Group Records

PlanExplode Record

A *PlanExplode* record contains the fields described in the following sections.

cellInstId

The *cellInstId* field contains the object ID of the cell instance that the *PlanExplode* record references.

instArray

This field is currently designated as internal to Synopsys applications.

numberOfInstance

This field is currently designated as internal to Synopsys applications.

flag

The *flag* field is internal to Synopsys applications.

PlanGroup Record

A *PlanGroup* record contains the fields described in the following sections.

bbox

See “[bbox](#)” on page [A-4](#) for definition.

cellInstGroupId

This field is currently designated as internal to Synopsys applications.

name

The *name* field contains the user specified text string identifying this record.

numberOfCellRow

The *numberOfCellRow* field contains the number of CellRow indicated within the *PlanGroup*.

planStatus

The *planStatus* field has 0-4 bits

- Bit0*axcGrpCellPlaced*
- Bit1*axcGrpInvisible*
- Bit2*axcGrpStandardCellMoveDefered*
- Bit3*axcGrpMacroCellMoveDefered*
- Bit4*axcGrpPlacementFixed*

targetHundredthOfPercent

This field is currently designated as internal to Synopsys applications.

type

A *type* field comprises two *PlanGroup* types.

- *SoftPlanGroup*
- *HardPlanGroup*

PlanGroupPin Record

A *PlanGroupPin* record contains the fields described in the following sections.

dataType

This field is currently designated as internal to Synopsys applications.

planGroupId

The *planGroupId* field contains the *planGroup* objectID that is attached to the net.

bbox

See “[bbox](#)” on page A-4 for definition.

layer

The *layer* field indicates how many layers are in the *PlanGroupPin* record.

netId

The *netId* field contains the object ID for the net to which this *PlanGroupPin* record is attached.

statusFlag

The *StatusFlag* field is internal to Synopsys applications. It has 0-4 bits

- Bit0*axcGrpCellPlaced*
- Bit1*axcGrpInvisible*
- Bit2*axcGrpStandardCellMoveDefered*
- Bit3*axcGrpMacroCellMoveDefered*
- Bit4*axcGrpPlacementFixed*

