

Milkyway™

Environment C-API

Reference Manual

Version A-2007.12, December 2008

Comments?

Send comments on the documentation by going to <http://solvnnet.synopsys.com>, then clicking "Enter a Call to the Support Center."

SYNOPSYS®

Copyright Notice and Proprietary Information

Copyright © 2008 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, Astro, Behavior Extracting Synthesis Technology, Cadabra, CATS, Certify, Design Compiler, DesignWare, Formality, HDL Analyst, HSPICE, Identify, iN-Phase, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Physical Compiler, PrimeTime, SCOPE, Simply Better Results, SiVL, SNUG, SolvNet, Synplicity, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, the Synplicity logo, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, BEST, Columbia, Columbia-CE, Confirma, Cosmos, CosmosLE, CosmosScope, CRITIC, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, Direct Silicon Access, Discovery, Eclipse, Encore, EPIC, Galaxy, Galaxy Custom Designer, HANEX, HAPS, HapsTrak, HDL Compiler, Hercules, Hierarchical Optimization Technology, High-performance ASIC Prototyping System, HSIM, HSIM^{plus}, i-Virtual Stepper, IICE, in-Sync, iN-Tandem, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Magellan, Mars, Mars-Rail, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, MultiPoint, Physical Analyst, Planet, Planet-PL, Polaris, Power Compiler, Raphael, Saturn, Scirocco, Scirocco-i, Star-RCXT, Star-SimXT, System Compiler, System Designer, Taurus, TotalRecall, TSUPREM-4, VCS Express, VCSi, VHDL Compiler, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (SM)

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.

ARM and AMBA are registered trademarks of ARM Limited.

Saber is a registered trademark of SabreMark Limited Partnership and is used under license.

All other product or company names may be trademarks of their respective owners.

Contents

What's New in This Release	xxviii
About This Manual	xxviii
Customer Support.	xxxix
1. Using the Milkyway C-API	
Overview	1-3
Milkyway Database Overview	1-3
Database Object Relationships	1-4
Object IDs.	1-4
Database Access Mechanisms	1-5
Dynamic C-API Programs.	1-5
Stand-Alone C-API Programs.	1-5
Tcl and Scheme Extensions	1-6
Database Access Methodologies	1-6
Tcl Methodology	1-6
Scheme Methodology.	1-7
Dynamic Methodology	1-8
Stand-Alone Methodology	1-9
Milkyway C-API Components	1-10
How to Describe C-API Functions	1-11
Detailed Format	1-11
Prototype-only Format	1-11
Setup Information	1-12
Installation Requirements	1-12
Dynamic Version.	1-12

Stand-Alone Version	1-12
Database Initialization for Stand-Alone Programs	1-13
Compilation Scripts for Stand-Alone Executables	1-13
C Program Examples	1-14
Cell Access Examples	1-15
Query Hierarchical Object Example.	1-20
Create Path Dynamic C-API Example	1-25
Creating and Registering a Function	1-26
Sample Open Cell Function.	1-27
Interpreting the Sample Program Subroutines	1-28
Registering a Function.	1-30
Loading Modules	1-32
C-API Function Return Codes	1-33
2. Working With Milkyway Database Objects	
Working With Libraries	2-2
Naming Libraries	2-2
Accessing Libraries	2-2
Specifying A Library ID	2-3
Interpreting the Variable ID Example	2-3
Recovering a Variable ID	2-4
Functions Used on Libraries	2-4
Working With Cells	2-5
Naming Cells	2-5
Naming Conventions	2-6
Version Numbering	2-6
Accessing Cells	2-7
Specifying a Cell ID	2-7
Interpreting the Cell ID Example.	2-8
Specifying a Cell ID	2-8
Functions Used on Cells	2-8
3. Database Objects, Object Fields, and Field Accessor Functions	
Accessing Objects in Cells	3-2
Opening a Library	3-2
Opening a Cell.	3-2

Getting Objects From Cells	3-3
Getting Data Fields of Objects	3-4
Deleting Objects	3-5
Object Types and Object Fields	3-5
Object Types	3-5
Common Object Fields	3-8
Object Field Descriptions and Field Accessor Functions	3-9
BaseArray Object	3-9
Boundary Object	3-11
BPVContactArray Object	3-12
CEdge Object	3-12
CellEquivalentClass Object	3-13
CellInfo Object	3-14
CellInst Object	3-16
CellInstArray Object	3-18
CellInstMaster Object	3-21
CellRegion Object	3-23
CellRow Object	3-24
CLFRecord Object	3-24
ClkOptData Object	3-25
Contact Object	3-26
ContactArray Object	3-28
DelayTable Object	3-29
ErrorComponent Object	3-30
ErrorObj Object	3-31
ErrorType Object	3-33
ExpCellInst Object	3-34
ExpPinInst Object	3-35
ExpPortInst Object	3-36

ExtraPin Object	3-36
ExtraPinInstMaster Object	3-37
ExtraPort Object	3-38
ExtraPortInst Object	3-39
ExtraPortInstMaster Object	3-40
Glink Object	3-40
Group Object	3-42
HierCellInst Object	3-42
HierCellInstMaster Object	3-43
HierPortInst Object	3-44
HierNet Object	3-45
HierPortVectorInfo Object	3-46
HorizontalWire Object	3-47
IGRPath Object	3-48
InstPower Object	3-48
LEdge Object	3-49
LibInfo Object	3-49
PreRouteSegment Object	3-50
Net Object	3-50
Path Object	3-54
Pin Object	3-55
PinAccessEdge Object	3-59
PinInstMaster Object	3-60
PlanGroup Object	3-62
Polygon Object	3-64
Port Object	3-65
PortCap Object	3-67
PortFunc Object	3-67

PortInst Object	3-68
PortInstMaster Object	3-69
PortToPortDelay Object	3-70
PortToPortTable Object	3-71
Rectangle Object	3-73
RectArray Object	3-74
REdge Object	3-74
RegionBlockage Object	3-75
RLCCoupleNet Object	3-75
RLCNet Object	3-77
RLCNode Object	3-79
RLCPort Object	3-79
RouteGuide Object	3-80
Schematic Object Functions	3-81
SchArc Object	3-82
SchBundle Object	3-83
SchCircle Object	3-84
SchCustom Object	3-85
SchGlobalNet Object	3-86
SchJunction Object	3-87
SchLines Object	3-87
SchNameAnnotation Object	3-88
SchPinInst Object	3-89
SchPin Object	3-90
SchPolygon Object	3-91
SchPort Object	3-91
SchProperty Object	3-92
SchRectangle Object	3-93
SchRipper Object	3-94
SchSegment Object	3-94
SchSubnet Object	3-95
SchText Object	3-96

Text Object	3-97
TilePattern Object	3-98
VerticalWire Object	3-99
ViaRegion Object	3-100
WireDirection Object	3-102
WireMaster Object	3-102
WireTrack Object	3-104
Common Object Fields	3-105
direction	3-105
accessDirection	3-106
cellFlag	3-106
recType	3-108
routeType	3-108
statusFlag	3-109
tableType	3-110
transformCode	3-111

4. Database Functions

MWXDb_Abort_Cell	4-2
MWXDb_Add_CellInstance_ToCellRegion	4-3
MWXDb_Add_CellInst_ToPlanGroup_ByName	4-4
MWXDb_Add_NetSubType	4-5
MWXDb_Add_NetSubType_ByName	4-7
MWXDb_Add_Object_ToGroup	4-8
MWXDb_Add_RefLib	4-9
MWXDb_Archive_CellToLib	4-11
MWXDb_Archive_HierDesign	4-12
MWXDb_Begin_ReadLib	4-13
MWXDb_Begin_WriteLib	4-15
MWXDb_Change_Cell_AccessMode	4-15

MWXDb_Check_Cell_Exist	4-16
MWXDb_Check_Cell_ExistInLibrary	4-17
MWXDb_Check_IdenticalCell.	4-19
MWXDb_Check_RefLibControl	4-19
MWXDb_Cleanup_ShapeStruct	4-20
MWXDb_Clear_CellMustJoinPin	4-21
MWXDb_Clear_RefLibControl	4-22
MWXDb_Close_Cell.	4-23
MWXDb_Close_Cell_And_NoSave	4-25
MWXDb_Close_Cell_And_Save.	4-26
MWXDb_Close_Lib	4-28
MWXDb_Connect.	4-29
MWXDb_Connect_PortInstance_ByName	4-30
MWXDb_Connect_PortWithPin	4-31
MWXDb_Convert_ObjectsToPath.	4-32
MWXDb_Convert_PathToPolygon	4-33
MWXDb_Convert_PathToRectangles.	4-35
MWXDb_Convert_Point_ToDBPoint	4-37
MWXDb_Convert_PolygonToRectangles	4-38
MWXDb_Convert_RectangleToPolygon	4-39
MWXDb_Copy_Cell	4-41
MWXDb_Copy_CellFromAnotherLib	4-43
MWXDb_Copy_CellToAnotherLibDisk	4-45
MWXDb_Copy_CellWithTIM.	4-46
MWXDb_Copy_Lib	4-47
MWXDb_Create_Cell	4-49
MWXDb_Create_Cell_Property	4-50
MWXDb_Create_CellAttachFile	4-52

MWXdB_Create_CellBoundary	4-53
MWXdB_Create_CellInstance	4-54
MWXdB_Create_CellInst_WithoutCheckStatus	4-56
MWXdB_Create_CellRegion	4-58
MWXdB_Create_CellRow	4-59
MWXdB_Create_ContactCode	4-61
MWXdB_Create_ErrorComponents	4-62
MWXdB_Create_ErrorObj	4-64
MWXdB_Create_ErrorType	4-66
MWXdB_Create_Group	4-67
MWXdB_Create_Horizontal_Wire	4-69
MWXdB_Create_Lib	4-70
MWXdB_Create_LibAttachFile	4-72
MWXdB_Create_Net	4-73
MWXdB_Create_ParameterSet	4-74
MWXdB_Create_Path	4-75
MWXdB_Create_Pin	4-78
MWXdB_Create_PinWithPortName	4-80
MWXdB_Create_PlanGroup	4-82
MWXdB_Create_Polygon	4-84
MWXdB_Create_Port	4-86
MWXdB_Create_Property	4-88
MWXdB_Create_Rectangle	4-90
MWXdB_Create_TdfPad	4-92
MWXdB_Create_Text	4-95
MWXdB_Create_Wire_Master	4-97
MWXdB_Delete_Cell	4-99
MWXdB_Delete_CellAttachFile	4-99

MWXDb_Delete_CellProperty_ByName	4-100
MWXDb_Delete_LibAttachFile	4-101
MWXDb_Delete_Object	4-102
MWXDb_Delete_ObjectArray	4-103
MWXDb_Delete_Objects_ByLayer	4-104
MWXDb_Delete_Pin	4-105
MWXDb_Delete_Property	4-106
MWXDb_Delete_Property_ByName	4-107
MWXDb_Delete_RefLib_ByName	4-108
MWXDb_Disconnect	4-109
MWXDb_Disconnect_PortInstance_ByName	4-110
MWXDb_Display_DiodePinInstMaster_FromDiodePortInstMaster	4-111
MWXDb_Display_DiodePortInst	4-112
MWXDb_Display_DiodePortInst_FromCellInst	4-113
MWXDb_Display_DiodePortInst_FromNet	4-114
MWXDb_Display_DiodePortInstMaster	4-115
MWXDb_Display_DiodePortInstMaster_FromDiodeCellInstMaster	4-116
MWXDb_Dump_TechFile	4-117
MWXDb_End_ReadLib	4-119
MWXDb_End_WriteLib	4-120
MWXDb_Fetch_Object	4-122
MWXDb_Get_AccessEdgeFromPin	4-123
MWXDb_Get_AllCellProperties	4-124
MWXDb_Get_AllObjectProperties	4-125
MWXDb_Get_AllPin_ByName	4-126
MWXDb_Get_CaseSensitive	4-127
MWXDb_Get_Cell_AccessMode	4-128
MWXDb_Get_CellAttachedFiles	4-129

MWXDb_Get_CellBBox	4-131
MWXDb_Get_CellBoundary	4-132
MWXDb_Get_CellFullName	4-133
MWXDb_Get_CellFullPath	4-135
MWXDb_Get_CellInfo_cellNameAndView	4-136
MWXDb_Get_CellInfo_fullName	4-137
MWXDb_Get_CellInstance_ByName	4-137
MWXDb_Get_CellInstanceBBox	4-138
MWXDb_Get_CellInstanceOrArray_ByName	4-139
MWXDb_Get_CellInstCellType_Bit	4-142
MWXDb_Get_CellInstFromCellInstMaster	4-143
MWXDb_Get_CellLatestVersion	4-144
MWXDb_Get_CellName_ByCellId	4-145
MWXDb_Get_CellNames_ByView	4-147
MWXDb_Get_CellProperty_ByName	4-148
MWXDb_Get_CellType_Bit	4-150
MWXDb_Get_ColorIdByName	4-151
MWXDb_Get_ConnectedObject_ById	4-152
MWXDb_Get_Constants	4-153
MWXDb_Get_ContactCode_ByName	4-154
MWXDb_Get_ContactCodeIdx_ByName	4-156
MWXDb_Get_ContactCodeNames	4-157
MWXDb_Get_ContactGeom	4-158
MWXDb_Get_ContactName_ByIndex	4-160
MWXDb_Get_CurrentLibId	4-161
MWXDb_Get_DBUPerUDistU	4-162
MWXDb_Get_DefaultContactByMetalLayers	4-163
MWXDb_Get_DefaultContactByViaLayer	4-164

MWXDb_Get_EditCell	4-165
MWXDb_Get_FatContactByMaskName	4-166
MWXDb_Get_Group_ByName	4-167
MWXDb_Get_GroupMember	4-169
MWXDb_Get_HierRefLibMode	4-170
MWXDb_Get_InputData	4-170
MWXDb_Get_Instances_ByBBoxSearch	4-171
MWXDb_Get_LayerId_ByName	4-172
MWXDb_Get_LayerId_ByNumber	4-173
MWXDb_Get_LayerName	4-174
MWXDb_Get_LayerNames	4-175
MWXDb_Get_LibAttachedFiles	4-177
MWXDb_Get_LibId_ByCellId	4-178
MWXDb_Get_LibId_ByLibName	4-179
MWXDb_Get_LibHierDelimiter	4-181
MWXDb_Get_LibName_ById	4-182
MWXDb_Get_LibViewNames	4-184
MWXDb_Get_LogicModelSubType	4-185
MWXDb_Get_MaskNames	4-186
MWXDb_Get_Net_ByName	4-188
MWXDb_Get_NetComponents	4-188
MWXDb_Get_NetPort	4-190
MWXDb_Get_NetPortInst	4-191
MWXDb_Get_NetWiring	4-192
MWXDb_Get_Objects_ByBBox	4-193
MWXDb_Get_Objects_ByBBoxSearch	4-195
MWXDb_Get_Objects_ByType	4-196
MWXDb_Get_Objects_ByTypeAndLayer	4-198

MWXDb_Get_ObjectsCount	4-199
MWXDb_Get_ObjectsType	4-200
MWXDb_Get_OwnerLibId_ByCellName	4-200
MWXDb_Get_Pin_ByName	4-203
MWXDb_Get_PinFromPort	4-204
MWXDb_Get_PinInstMaster_numberOfPoint	4-205
MWXDb_Get_PinNet_ByName	4-205
MWXDb_Get_PlanGroup_ByName	4-206
MWXDb_Get_Port_ByName	4-207
MWXDb_Get_PortInstFromCellInst	4-208
MWXDb_Get_PrBoundary	4-209
MWXDb_Get_Property	4-210
MWXDb_Get_Property_ByName	4-212
MWXDb_Get_RefLib	4-214
MWXDb_Get_RefLibPathNames	4-215
MWXDb_Get_RegionPort_ByName	4-218
MWXDb_Get_RegionPort_Direction	4-218
MWXDb_Get_SelectedSet	4-219
MWXDb_Get_ViaRegionFromPort	4-221
MWXDb_Get_WireDirection_layer	4-222
MWXDb_Get_WireDirection_preferedDirection	4-223
MWXDb_Get_WireMasterId	4-223
MWXDb_Get_WireTrack_bbox	4-224
MWXDb_Get_WireTrack_layer	4-225
MWXDb_Get_WireTrack_startTrack	4-226
MWXDb_Get_WireTrack_times	4-227
MWXDb_Get_WireTrack_trackDirection	4-228
MWXDb_Get_WireTrack_trackStep	4-229

MWXDb_Get_WireTrack_type	4-230
MWXDb_Is_CaseSensitive.	4-231
MWXDb_Is_NetSubTypeCoreAndPad	4-233
MWXDb_Is_NetSubTypeCoreOnly	4-233
MWXDb_Is_NetSubTypeCTSNet	4-234
MWXDb_Is_NetSubTypeCTSRootNet	4-235
MWXDb_Is_NetSubTypeNonCTSNet.	4-236
MWXDb_Is_NetSubTypePadOnly	4-237
MWXDb_Is_Object_InGroup	4-238
MWXDb_Iter_{Object}To{Attachment}	4-239
MWXDb_Iter_LibCellViewPair	4-243
MWXDb_Iter_Next	4-244
MWXDb_Iter_Next_LibCellViewPair.	4-245
MWXDb_Iter_ObjectByType.	4-248
MWXDb_Iter_Quit.	4-250
MWXDb_Iter_Quit_LibCellViewPair	4-251
MWXDb_Merge_Objects	4-251
MWXDb_omwxInit	4-252
MWXDb_Open_Cell	4-253
MWXDb_Open_CellAttachFile	4-254
MWXDb_Open_Cell_ByLibId	4-255
MWXDb_Open_Lib.	4-258
MWXDb_Open_LibAttachFile.	4-260
MWXDb_Open_Library_ByAccessMode	4-261
MWXDb_Open_RefLibAttachFile	4-263
MWXDb_Output_ClickedPoint	4-264
MWXDb_Place_CellInstanceInLegalLoc	4-265
MWXDb_Purge_AllPlanGroup	4-267

MWXDb_Purge_AllSingleNet	4-267
MWXDb_Purge_AllTdfPads	4-268
MWXDb_Purge_CellInstance_ByName	4-269
MWXDb_Purge_CellInstMaster	4-270
MWXDb_Purge_Group_ByName	4-271
MWXDb_Purge_Net	4-272
MWXDb_Purge_SingleObjectType	4-273
MWXDb_Rebuild_CellWithVersion	4-275
MWXDb_Rebuild_Library	4-276
MWXDb_Rebuild_Library_WithAccessCheck	4-278
MWXDb_Recompute_CellBBox	4-280
MWXDb_Remove_Object_FromGroup	4-281
MWXDb_Rename_Cell	4-282
MWXDb_Rename_Lib	4-283
MWXDb_Replace_CellInst	4-285
MWXDb_Replace_CellTextChar	4-286
MWXDb_Replace_CellTextString	4-287
MWXDb_Replace_ChildCell	4-288
MWXDb_Replace_RefLib_ByName	4-289
MWXDb_Replace_TechInfo	4-290
MWXDb_Replace_TextChar	4-291
MWXDb_Replace_TextString	4-292
MWXDb_Restore_Pin	4-293
MWXDb_RollBack_Cell	4-295
MWXDb_Save_Cell	4-295
MWXDb_Set_CaseSensitive	4-296
MWXDb_Set_CellInstanceFixStatus_ByName	4-297
MWXDb_Set_CellInstancePlacement_ByName	4-298

MWXDb_Set_CellLayoutSubType	4-300
MWXDb_Set_CellLModelSubType	4-301
MWXDb_Set_CellType	4-301
MWXDb_Set_CellType_Bit	4-302
MWXDb_Set_ErrorObjStatus	4-303
MWXDb_Set_ErrorTypeStatus	4-304
MWXDb_Set_GetBBoxCallBack	4-305
MWXDb_Set_GetLineCallBack	4-307
MWXDb_Set_GetPointCallBack	4-309
MWXDb_Set_HierRefLibMode	4-311
MWXDb_Set_IOPad	4-312
MWXDb_Set_NetSubType	4-313
MWXDb_Set_NetSubType_ByName	4-314
MWXDb_Set_NetType	4-315
MWXDb_Set_NetType_ByName	4-316
MWXDb_Set_RefControlFileMode	4-317
MWXDb_Set_RefLibControl	4-318
MWXDb_Set_StdCell	4-319
MWXDb_Set_TdfLoadCapacitance	4-320
MWXDb_Set_TdfLoadCapacitanceInDouble	4-321
MWXDb_Set_TdfSrcCapacitanceList	4-321
MWXDb_Set_TdfSrcCapacitanceListInDouble	4-322
MWXDb_Set_TdfSrcResistanceList	4-323
MWXDb_Set_TdfSrcResistanceListInDouble	4-324
MWXDb_Set_TechFileName	4-325
MWXDb_Show_DiodePortConn	4-326
MWXDb_Startup_ShapeStruct	4-327
MWXDb_Store_CellAttachFile	4-328

MWXDb_Store_LibAttachFile	4-329
MWXDb_Transform_Object	4-330
MWXDb_Truncate_CellText	4-331
MWXDb_Truncate_Text	4-332
MWXDb_Update_CellProperty_ByName	4-333
MWXDb_Update_Property	4-335
MWXDb_Update_Property_ByName	4-337

5. Technology File Data Access Functions

MWXDb_Get_ColorName	5-2
MWXDb_Get_ColorRGB	5-2
MWXDb_Get_ColorRGBDefined	5-3
MWXDb_Get_DesignRuleDiffNetMinSpacing	5-4
MWXDb_Get_DesignRuleMinEnclosure	5-5
MWXDb_Get_DesignRuleMinSpacing	5-6
MWXDb_Get_DesignRuleStackable	5-7
MWXDb_Get_FringeCapInfo	5-8
MWXDb_Get_LayerBorderStyleName	5-9
MWXDb_Get_LayerCapacitance	5-10
MWXDb_Get_LayerColorName	5-13
MWXDb_Get_LayerDisplayAttr	5-14
MWXDb_Get_LayerEtchFactor	5-15
MWXDb_Get_LayerFatWireRules	5-17
MWXDb_Get_LayerFringeCap	5-18
MWXDb_Get_LayerHeightFromSub	5-19
MWXDb_Get_LayerInductance	5-20
MWXDb_Get_LayerLayoutAttr	5-21
MWXDb_Get_LayerLineStyleName	5-22

MWXDb_Get_LayerName_FromNumber	5-23
MWXDb_Get_LayerNumber_FromName	5-24
MWXDb_Get_LayerPhysicalAttr	5-25
MWXDb_Get_LayerResistance	5-26
MWXDb_Get_LayerStippleName	5-27
MWXDb_Get_LayerStubRouting	5-28
MWXDb_Get_LayerThickness	5-29
MWXDb_Get_TechCapacitance	5-30
MWXDb_Get_TechContactLayer	5-31
MWXDb_Get_TechContactSize	5-32
MWXDb_Get_TechCurrent	5-33
MWXDb_Get_TechDielectric	5-34
MWXDb_Get_TechInductance	5-35
MWXDb_Get_TechName	5-36
MWXDb_Get_TechPower	5-37
MWXDb_Get_TechResistance	5-38
MWXDb_Get_TechResolution	5-39
MWXDb_Get_TechRowAbutable	5-40
MWXDb_Get_TechRowSpacing	5-41
MWXDb_Get_TechTime	5-42
MWXDb_Get_TechVoltage	5-43

6. Scheme I/O and Boolean Functions

MWX_Log	6-2
MWX_Printf	6-2
MWXDb_BooleanAnd_Objects	6-3
MWXDb_BooleanNot_Objects	6-4
MWXDb_BooleanOr_Objects	6-5

MWXDb_BooleanXor_Objects	6-6

7. Scheme Interpreter Functions

Classification of Scheme Interpreter Functions	7-2
Scheme Function Descriptions	7-3
MWXSc_Car	7-3
MWXSc_Cdr	7-4
MWXSc_Evaluate	7-4
MWXSc_Get_BooleanSCM	7-5
MWXSc_Get_DBIdSCM	7-6
MWXSc_Get_DoubleSCM	7-7
MWXSc_Get_IntegerSCM	7-8
MWXSc_Get_StringSCM	7-8
MWXSc_Get_Symbol	7-9
MWXSc_Is_BooleanSCM	7-10
MWXSc_Is_DoubleSCM	7-10
MWXSc_Is_DBIdSCM	7-11
MWXSc_Is_IntegerSCM	7-11
MWXSc_Is_ListSCM	7-12
MWXSc_Is_NullSCM	7-13
MWXSc_Is_StringSCM	7-13
MWXSc_Make_BooleanSCM	7-14
MWXSc_Make_DBIdSCM	7-14
MWXSc_Make_DoubleSCM	7-15
MWXSc_Make_IntegerSCM	7-15
MWXSc_Make_ListSCM	7-16
MWXSc_Make_StringSCM	7-17
MWXSc_Register_Function	7-17

MWXSc_Scanf	7-18

8. Place and Route Preparation Functions

MWXDb_Add_InstGroup_ToScanChain	8-2
MWXDb_Add_PartialScan_ToScanChain	8-3
MWXDb_Assign_VarRouteRule_ById	8-4
MWXDb_Assign_VarRouteRule_ByNetName	8-5
MWXDb_Attach_ChildCellInstToHierCellInst	8-7
MWXDb_Attach_ChildHierCellInstToHierCellInst	8-8
MWXDb_Attach_HierPortInstMasterToHierCellInstMaster	8-9
MWXDb_Attach_HierPortInstToHierCellInst	8-10
MWXDb_Attach_HierPortInstToHierNet	8-11
MWXDb_Attach_PortInstToHierNet	8-13
MWXDb_Clear_CellEEQClass	8-14
MWXDb_Clear_CellEQClass	8-15
MWXDb_Clear_CellLEQClass	8-16
MWXDb_Create_BaseArray	8-17
MWXDb_Create_BaseArrayRow	8-18
MWXDb_Create_CellEEQClass	8-19
MWXDb_Create_CellInst_IgnoreECO	8-20
MWXDb_Create_CellLEQClass	8-21
MWXDb_Create_Contact_WithRoutType	8-23
MWXDb_Create_ContactArray_WithRoutType	8-24
MWXDb_Create_CoreBaseArray	8-27
MWXDb_Create_Glink	8-28
MWXDb_Create_HierCellInst	8-29
MWXDb_Create_HierCellInstMaster	8-31
MWXDb_Create_HierNet	8-32

MWXdB_Create_HierPortInst.	8-33
MWXdB_Create_HierPortInstMaster	8-36
MWXdB_Create_HierPortVectorInfo.	8-37
MWXdB_Create_PcellDefinition.	8-39
MWXdB_Create_PcellInstance	8-40
MWXdB_Create_Row.	8-41
MWXdB_Create_ScanChain	8-43
MWXdB_Create_SGEdge	8-43
MWXdB_Create_SGPrimaryNode	8-44
MWXdB_Create_TilePattern	8-46
MWXdB_Create_Track	8-47
MWXdB_Create_Vertical_Wire	8-50
MWXdB_Create_WireDirection	8-51
MWXdB_Define_VarRouteRule_Contact	8-53
MWXdB_Define_VarRouteRule_Spacing.	8-54
MWXdB_Define_VarRouteRule_TaperLevel	8-56
MWXdB_Define_VarRouteRule_Width.	8-57
MWXdB_Delete_AllRouteGuides	8-59
MWXdB_Delete_AllVarRouteRules	8-59
MWXdB_Delete_CellInstance_FromCellRegion	8-60
MWXdB_Delete_PcellDefinition	8-61
MWXdB_Delete_ScanChain	8-62
MWXdB_Dump_AllVarRouteRules.	8-63
MWXdB_Force_UpdateCellHierarchy	8-64
MWXdB_Get_AllNetVarRouteRules.	8-65
MWXdB_Get_AllRouteGuides	8-67
MWXdB_Get_AllVarRouteRules.	8-68
MWXdB_Get_Cell_EEQClass	8-69

MWXDb_Get_Cell_LEQClass	8-70
MWXDb_Get_CellInst_PowerDelayScale.....	8-71
MWXDb_Get_CellInst_PowerId	8-72
MWXDb_Get_CellInst_PowerSupplyVoltage	8-73
MWXDb_Get_CellInst_PowerTemperature.....	8-74
MWXDb_Get_CellInst_PowerValue	8-75
MWXDb_Get_CellRegionIdFromCellInstId.....	8-76
MWXDb_Get_DataTypeName	8-77
MWXDb_Get_GlobalNet_ByNameMap	8-78
MWXDb_Get_NetLoad.....	8-79
MWXDb_Get_NetPowerId	8-80
MWXDb_Get_NetSwitchActivity.....	8-81
MWXDb_Get_RouteGuide	8-82
MWXDb_Get_RouteGuide_PropertyValueArray	8-83
MWXDb_Get_TopHierCellInst	8-85
MWXDb_Get_VarRouteRuleName_ByNetId	8-86
MWXDb_Get_VarRouteRuleSpacing_ByName	8-87
MWXDb_Get_VarRouteRuleTaperLevel_ByName.....	8-89
MWXDb_Get_VarRouteRuleWidth_ByName	8-91
MWXDb_Get_VarVarRouteRuleContact_ByName.....	8-93
MWXDb_Iter_Next_ExpPinInst.....	8-94
MWXDb_Iter_PortInstToExpPinInst	8-95
MWXDb_Iter_PortToTiming	8-99
MWXDb_LoadDesignSetupDone.....	8-101
MWXDb_Remove_InstGroup_FromScanChain	8-103
MWXDb_Reset_CellInstance_Traceable	8-104
MWXDb_Reset_Net_Traceable	8-104
MWXDb_Restore_HierDesign	8-105

MWXDb_Set_CellInst_PowerDelayScale	8-106
MWXDb_Set_CellInst_PowerSupplyVoltage	8-107
MWXDb_Set_CellInst_PowerTemperature	8-108
MWXDb_Set_CellInst_PowerValue	8-109
MWXDb_Set_CellInstanceECOStatus_ByName	8-110
MWXDb_Set_CellInstance_Traceable	8-111
MWXDb_Set_CellMustJoin2Pin	8-112
MWXDb_Set_CellMustJoinPinPattern	8-113
MWXDb_Set_HorizontalWire_routeType	8-114
MWXDb_Set_Net_Traceable	8-115
MWXDb_Set_Netload	8-116
MWXDb_Set_NetSwitchActivity	8-117
MWXDb_Set_ScanChain_DefaultPriority	8-118
MWXDb_Set_TopHierCellInst	8-119
MWXDb_Set_VerticalWire_routeType	8-120
MWXDb_Translate_MilkywayName_ToVerilogName	8-121
MWXDb_Translate_VerilogName_ToMilkywayName	8-122
MWXDb_Undelete_Object	8-124

9. PARA View Commands

Dynamic PARA View Pvx* Examples	9-2
MWXDb_Close_PvxDesign	9-16
MWXDb_Close_PvxRLCCoupleNetIteration	9-17
MWXDb_Close_PvxRLCNetIteration	9-17
MWXDb_Create_PvxRCNetworkForAllNets	9-18
MWXDb_Get_PvxCEdgeInfo	9-19
MWXDb_Get_PvxRLCCoupleNetAggNetInfo	9-21
MWXDb_Get_PvxRLCCoupleNetInfo	9-22

MWXDb_Get_PvxRLCEdgeInfo	9-23
MWXDb_Get_PvxRLCNet_ByName	9-25
MWXDb_Get_PvxRLCNetInfo	9-25
MWXDb_Get_PvxRLCNodeInfo	9-26
MWXDb_Get_PvxRLCPortInfo	9-28
MWXDb_Init_PvxDesign	9-29
MWXDb_Init_PvxRLCCoupleNetIteration.	9-30
MWXDb_Init_PvxRLCNetIteration	9-31
MWXDb_Iter_Next_PvxRLCCoupleNetMember_ByType.	9-32
MWXDb_Iter_Next_PvxRLCNetMember_ByType	9-33
MWXDb_Iter_PvxRLCNetMember_ByType	9-34
Dynamic PARA View Pvx* Example	9-35

Index

Preface

This preface includes the following sections:

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

What's New in This Release

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

Due to the enhanced database model of the Milkyway Environment version B-2008.09, the power and ground net rules are to be handled the same way signal net rules are handled.

For more information about the enhanced Milkyway database model, see the *BSD Compiler Release Notes*,

1. Go to <http://solvnet.synopsys.com/ReleaseNotes>. (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.)
2. Click Milkyway, then click the release you want in the list that appears at the bottom.

About This Manual

This reference manual provides information about using the Milkyway Database Access C-API. This C-API provides a solution for highly algorithmic applications or when a specific data structure is required.

Audience

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

Using Swish-e With Physical Implementation Online Help

The Swish-e program, which is provided for online Help, is not “Licensed Software” or “Licensed Product” in your license agreement with Synopsys. The Swish-e program is subject to the license provisions found at <http://swish-e.org>, where you can also find the source code for Swish-e.

Related Publications

For additional information about the *Milkyway Environment C-API 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/DocsOnWeb>
- 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 documentation for the following related Synopsys products:

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

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
<code>Courier</code>	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 <i>low medium high</i> (This example indicates that you can enter one of three possible values for an option: <i>low</i> , <i>medium</i> , or <i>high</i> .)
—	Connects terms that are read as a single term by the system, such as <i>set_annotated_delay</i>
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 HELP in the top-right menu bar or in the footer.

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 the Milkyway C-API

The Milkyway C-API tool is one of the extension mechanisms for tools based on Milkyway. The tool is used to annotate or post-process a database. It cannot be used to create a database from scratch.

Note:

IC Compiler supports only the Tcl scripting language. If your database is eventually going to be used in IC Compiler, it is recommended that you use Tcl instead of C-API or scheme to work with the database.

Synopsys provides the following versions of the C-API tool:

- Dynamic, which lets you write applications as C functions that are dynamically linked into the Milkyway database environment and accessed by Scheme commands. In this case, all existing Milkyway commands, including graphic display, tool commands, and file readers and writers are available using a mechanism to invoke Scheme from the C-API.
- Stand-alone, which lets you write applications that run independent of the Milkyway environment and GUI.

For more information about these versions, see [“Database Access Mechanisms” on page 1-5](#).

This chapter contains the following sections:

- [Overview](#)
- [Setup Information](#)

- [C Program Examples](#)
- [Creating and Registering a Function](#)

Overview

This section includes the following topics:

- [Milkyway Database Overview](#)
- [Database Access Mechanisms](#)
- [Database Access Methodologies](#)

Milkyway Database Overview

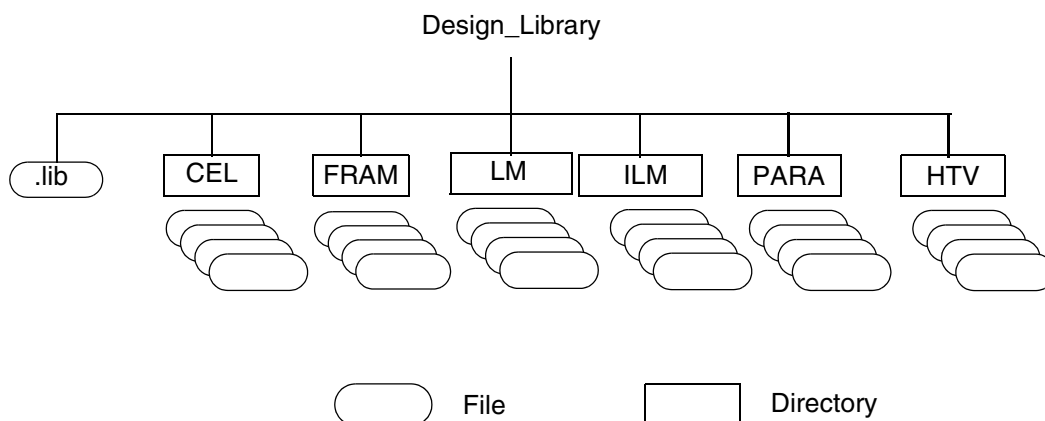
The top level Milkyway object is a library, which is implemented as a directory in the UNIX file system. Additional status information is contained in a file within the Library directory (which is shown as “lib” in [Figure 1-1](#)).

There are two library types: those containing a particular user design and those referenced by designs. For more information about Milkyway libraries, see [Chapter 2, “Working With Libraries.”](#)

A Library contains cells, which are implemented as file system directories for each view type and as files for the data itself. For more information about cells in Milkyway libraries, see [Chapter 2, “Working With Cells.”](#)

[Figure 1-1](#) shows an overview of the files that make up a Milkyway database plus external formats that must be imported and exported.

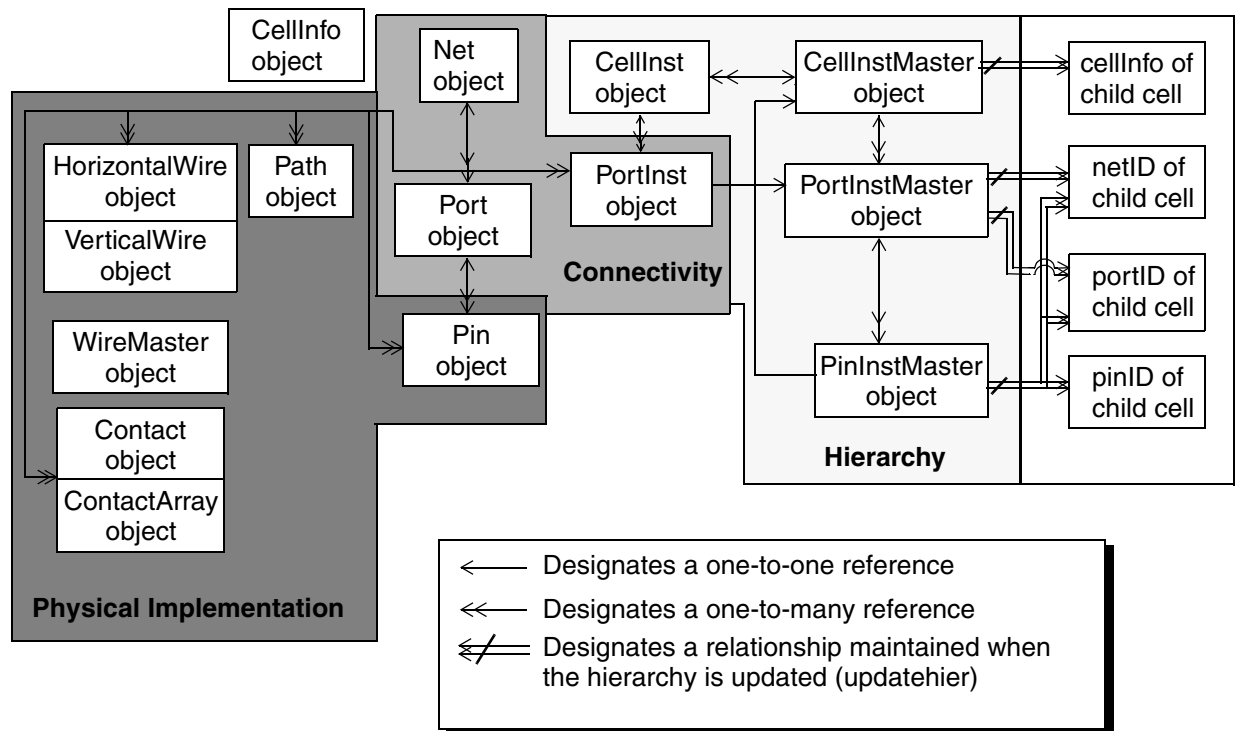
Figure 1-1 Milkyway File Structure



Database Object Relationships

Figure 1-2 shows some of the main objects in Milkyway and also illustrates some of the relationships between these objects.

Figure 1-2 Milkyway Objects



Object IDs

All objects in the Milkyway database are manipulated by *handles* known as Object IDs. There are three basic types of Object IDs:

- Library IDs, which are not persistent and are not context sensitive. They are assigned when the library is opened.
- Cell IDs, which are also not persistent and are not context sensitive. In particular they do not require the context of a Library ID to be unique—every cell opened is assigned a non-persistent ID unique within a particular session of database use.
- All other object IDs, are persistent and require the context of either a cell ID or a library ID to be understood by the database. This is because more than one cell or library may have an object with the same database ID value.

When a library or cell is opened, Milkyway assigns it a unique, temporary database ID that is dependent upon the order in which the library or cell is accessed. This ID is only valid until the cell or library is closed.

All other IDs, unlike library or cell IDs, are persistent. During the creation of an object, Milkyway 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 a given session, then port A will continue to have that same exact ID in future sessions.

Caution!

Under certain modification operations, an ID may change. Furthermore, 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.

Database Access Mechanisms

Synopsys provides the following mechanisms for creating custom design flows and extending operations on Milkyway data:

- [Dynamic C-API Programs](#)
- [Stand-Alone C-API Programs](#)
- [Tcl and Scheme Extensions](#)

Dynamic C-API Programs

The dynamic C-API facilitates the use of Scheme scripts that access design data in Milkyway databases. This is done with functions written in C that are dynamically loaded into the system. These functions are then called from Scheme and, because of this, the dynamic C-API includes functions and data types that interface between Scheme and C.

For more information about developing C program functions, see [Chapter 7, “Scheme Interpreter Functions.”](#)

For more information about using the dynamic C-API, see [“Dynamic Methodology.”](#)

Stand-Alone C-API Programs

The Milkyway C-API enables you to write stand-alone programs that access, modify, or create design data in files in Milkyway databases. The C-API can also be used to add to an existing program the ability to read and write Milkyway data.

The stand-alone C-API is based on and provides the same data access functions as the dynamic C-API. Both provide a set of C functions that are very similar to database access functions available in Scheme in Synopsys Milkyway-based tools. However the usage model for the stand-alone Milkyway Database Access C-API is significantly different than the usage model for the dynamic Milkyway C-API.

The stand-alone C-API provides the exact same C functions for database and technology data access as in the dynamic C-API, but because the programs are not run from the Synopsys tool Scheme environment, the Scheme interface functions and data types are not needed.

For more information about developing C program functions, see [Chapter 7, “Scheme Interpreter Functions.”](#)

For more information about using the dynamic C-API, see [“Dynamic Methodology.”](#)

Tcl and Scheme Extensions

For information about using the Scheme Programming Language, see the *Milkyway Environment Extension Language Reference Manual*.

Database Access Methodologies

The following sections describe the following methodologies for accessing Milkyway databases:

- [Tcl Methodology](#)
- [Scheme Methodology](#)
- [Dynamic Methodology](#)
- [Stand-Alone Methodology](#)

Tcl Methodology

Milkyway allows you to access the database using Tcl commands. There are two categories of Tcl commands that access the Milkyway environment, Tcl-equivalent commands and Synopsys Tcl commands. These Tcl categories are described briefly below. For more information, see the *Milkyway Environment Extension Language Reference Manual*.

First, with the introduction of the Milkyway Tcl extension language, most existing Scheme commands are available as Tcl commands. These commands are referred to as *Tcl-equivalent (of Scheme) commands*. With Tcl, you cannot create C functions and register them as new Tcl commands in the same way that you can with Scheme commands.

The Tcl-equivalent commands have the same names and arguments as their Scheme counterparts. Because Scheme and Tcl differ, there are slight differences in syntax. To determine the syntax of a Tcl-equivalent command, see the “Tcl-Equivalent Command Guidelines” table in the *Milkyway Environment Extension Language Reference Manual*.

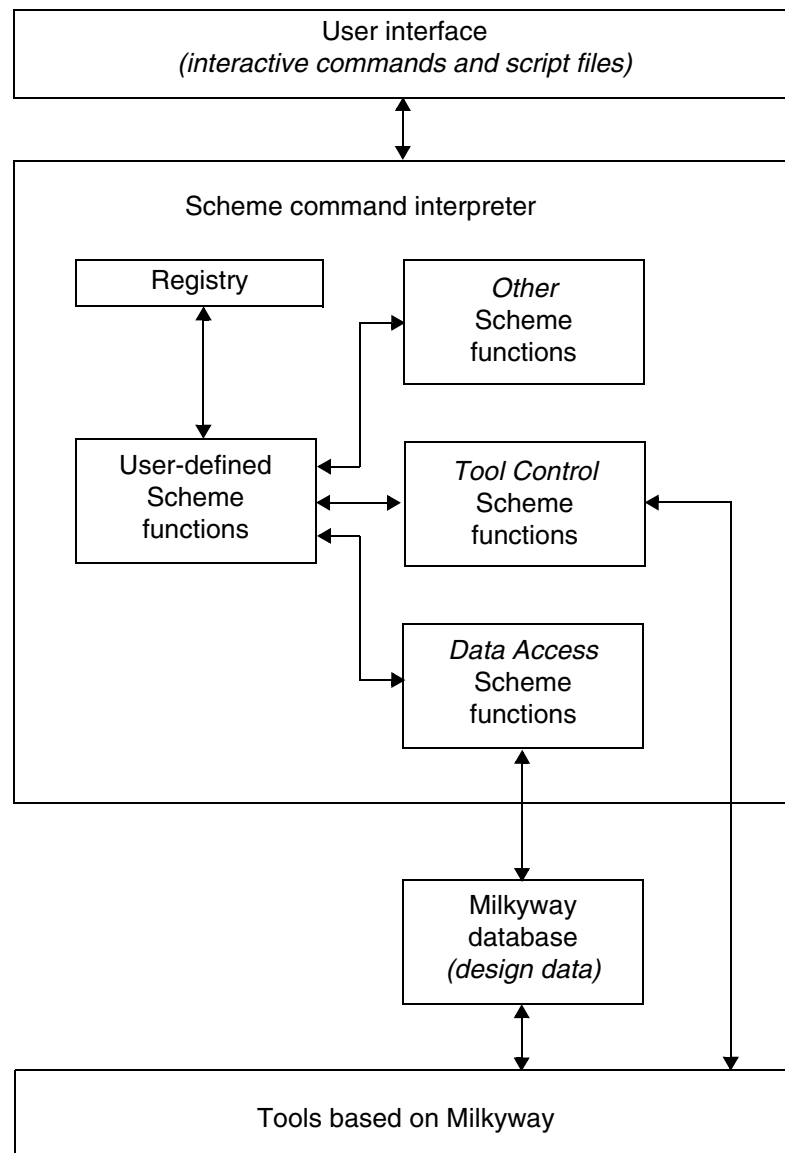
The second category of Tcl commands available in the Milkyway Environment are *Synopsys Tcl commands*. These commands are often compatibility across multiple tools. Synopsys Tcl commands can produce collections, as well as accept collections as arguments. With Synopsys Tcl, command abbreviation is also supported.

Scheme Methodology

Synopsys tools that run on Milkyway can use Scheme as a scripting and macro language. Scripts can be run from a file, or new functions can be defined in the registry using any Scheme functions, including those that access data and that control tool functions.

[Figure 1-3](#) shows that you can then invoke built-in and user-defined commands interactively or from scripts.

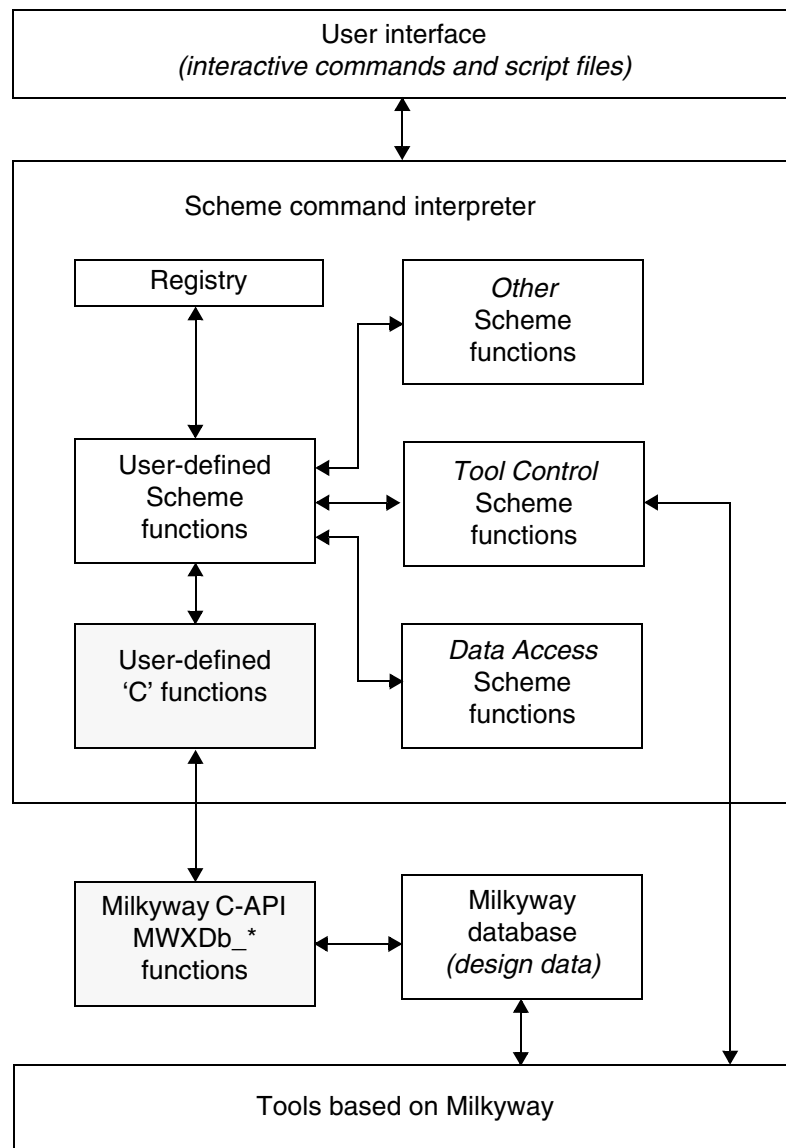
Figure 1-3 Scheme Methodology



Dynamic Methodology

When large amounts of data need to be processed, the processing speed can be enhanced by defining new functions in C using the dynamic C-API instead of Scheme. The new C functions define new Scheme functions that are entered into the registry and become new Scheme commands.

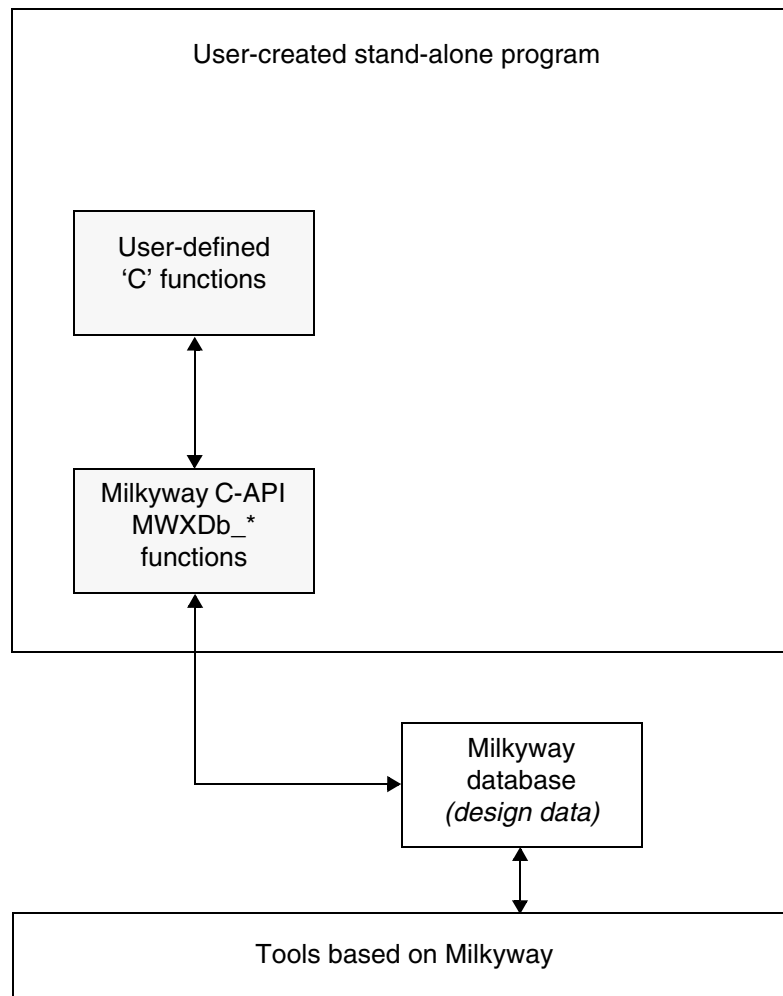
Figure 1-4 Dynamic Milkyway C-API Methodology



Stand-Alone Methodology

When access to Milkyway data is needed from a stand-alone program, the Milkyway Data Access C-API is used. The exact same database and technology C-API functions that were available before only via the Scheme registration process can be used in your own program.

Figure 1-5 Stand-Alone Milkyway Database Access C-API Methodology



Milkyway C-API Components

This document describes the components that make up the Milkyway C-API. The components are

- Database object definitions
- Object field definitions
- Object fields accessor functions

- Database functions
- Technology file data access functions
- Scheme I/O and Boolean functions
- Scheme interpreter functions
- Place and route preparation functions

Database objects, object fields, and object field accessor functions are described in [Chapter 3, “Database Objects, Object Fields, and Field Accessor Functions.”](#) The other types of functions are described in their own respective chapters.

How to Describe C-API Functions

You can describe the C-API functions in one of two ways:

- [Detailed Format](#)
- [Prototype-only Format](#)

Detailed Format

A detailed description format includes the following elements:

- Syntax
- Parameters
- Return
- Example

When applicable, the following additional elements are also included:

- See Also
- Scheme Equivalent

Prototype-only Format

The prototype-only format is used in conjunction with a large class of object field accessor functions (`MWXdB_Get_object_field`). These functions are used either to get or to set fields within database objects. All these functions work in a similar manner, and their usage is described in [Chapter 3, “Getting Data Fields of Objects.”](#) Only the prototypes for these accessor functions are shown. These prototypes follow descriptions of their corresponding database objects.

Setup Information

The following setup information describes the general process for installing Milkyway. This section includes the following:

- [Installation Requirements](#)

Installation Requirements

This section lists the installation requirements for the following versions:

- [Dynamic Version](#)
- [Stand-Alone Version](#)

Dynamic Version

The following are required to install the Milkyway C-API dynamic version:

- Milkyway C-API files
 - `mw_x.h` - header file that defines Milkyway data types and constants
 - `mw_x_proto.h` - header file with all of the original Milkyway C-API function signatures (prototypes)
 - `AvantAr` - program that prepares your code for loading into the Milkyway environment.
- Documentation
 - *Milkyway Environment C-API Reference Manual* (this manual)
 - README in PDF

Stand-Alone Version

The following are required to install the Milkyway C-API stand-alone version:

- Milkyway C-API files
 - `mw_x.h` - header file that defines Milkyway data types and constants
 - `mw_x_proto.h` - header file with all of the original Milkyway C-API function signatures (prototypes)
 - `omw_x_proto.h` - header file with stand-alone Milkyway C-API signatures (prototypes). Currently there is only one new function for the stand-alone version used to initialize Milkyway: `MWXDb_omw_x_init()`.

- libOMWX.a - link library with the Milkyway C-API functions.
- Documentation
 - *Milkyway Environment C-API Reference Manual* (this manual)
 - README in PDF

Database Initialization for Stand-Alone Programs

Before any operation can be performed on a Milkyway database by a stand-alone program (including opening a library or cell), it must first be initialized by using the `MWXDb_omwxInit` function. This function is described in [Chapter 4, “MWXDb_omwxInit.”](#)

Compilation Scripts for Stand-Alone Executables

This section contains simple example compilation scripts for creating stand-alone executables for the following platforms:

- [IA.32 Compilation Script](#)
- [SUN.32 Compilation Script](#)

All of these examples assume that the header files and the linkable library (libOMWX.a) are in the same directory as the script and the C program.

The installation for the C-API includes many examples. You can find these under an example directory initially as Example.tar.gz. When this tar file is uncompressed and expanded the Example directory contains a README file with instructions for building and running the examples. Within an src sub-directory there is a Makefile than can handle the various compute platforms requirements for compilation and linking.

IA.32 Compilation Script

The following shell script uses the GNU gcc compiler with the previous program example to create a stand-alone executable. This example works on Red Hat Enterprise Linux version 3 running on an x86 processor.

```
#!/bin/bash
echo ]----- COMPILING -----[
gcc \
  -DIA32_LINUX__2_4 \
  -I. \
  -I/usr/local/include \
  -I/usr/include \
  -L. \
  -L/usr/local/lib \
  -L/usr/lib \
  -L/usr/X11R6/lib \
  -g \
```

```

-o my_programs\
  my_example.c \
-lOMWX \
-lX11 \
-ldl \
-lnsl \
-lm \
echo ]----- $0 DONE.

```

SUN.32 Compilation Script

The following shell script uses the GNU gcc compiler with the previous program example to create a stand-alone executable. This example works on Sun Solaris 2.7 or Solaris 8 (SunOS 5.7 or SunOS 5.8). Note that the various -D parameters passed to the C compiler are required for correct operation.

```

#!/bin/sh
echo ]----- COMPILING -----[
gcc \
  -I. \
  -I/usr/local/include \
  -I/usr/include \
  -L. \
  -L/usr/local/lib \
  -L/usr/lib \
  -g \
  -o my_programs\
    my_example.c \
-lOMWX \
-lX11 \
-ldl \
-lm \
-lnsl \
-lsocket \
-lsunmath \
echo ]----- $0 DONE.

```

C Program Examples

The following sections include examples using the C-API.

- [Cell Access Examples](#)
- [Query Hierarchical Object Example](#)

Cell Access Examples

The following example programs use the C-API to open a library, open a cell, open the cell a second time (to get an already open error), then close the cell and exit. [Example 1-3](#) shows the output from running the example program.

[Example 1-1](#) shows how to use the dynamic version to open a cell.

Example 1-1 Accessing a Cell, Dynamic Version

```
#include<stdio.h>
#include "mwx.h"
#include "mwx_proto.h"

#ifdef __cplusplus
extern "C" {
    MWXSCM_t helloworld();
    void reg_helloworld();
}
#endif

MWXSCM_t helloworld (MWXSCM_t list)
{
    char *progDescrString = "MW-CAPI Example 1 -";
    static int caseSensitive = 0;
    char myLibName[256];
    MWXLibId_t myLibId;
    char myCellName[256];
    char myCellAccessModeChr = 'w';
    MWXCellId_t myCellId;
    int rc;

    /*-----
       GET THE SCHEME LIST
       -----*/
    MWXSc_Scanf(list, "ss",myLibName,myCellName);

    /*-----
       Getting Started with the Milkyway Dynamic C-API. This program opens a
       library, opens a cell, closes the cell, and closes the library.
       -----*/
    MWX_Printf("\n%s HELLO...\n", progDescrString);

    /*-----
       OPEN LIBRARY
       -----*/
    MWX_Printf("%s Opening LIBRARY %s USING MWXDb_Open_Lib
    ...\n",progDescrString,
        myLibName);

    rc = MWXDb_Open_Lib(myLibName,&myLibId);

    switch(rc) {
        case MWXSuccess:
            MWX_Printf("%s SUCCESSFUL : LIBRARY OPEN.\n", progDescrString);
```

```

        break;
    case MWXFail:
        MWX_Printf("%s *** ERROR - Library Open Failure\n",progDescrString);
        return(MWX_FALSE_SCM);
    case MWXNoLicense:
        MWX_Printf("%s *** ERROR - NO LICENSE\n", progDescrString);
        return(MWX_FALSE_SCM);
    default:
        MWX_Printf("%s *** UNKNOWN ERROR %d from
        MWXDb_Open_Lib\n",progDescrString, rc);
        return(MWX_FALSE_SCM);
}
MWX_Printf("%s --- OPENED LIBRARY ID is %d\n",progDescrString,myLibId);

/*-----
    OPEN CELL
    -----*/
MWX_Printf("%s Opening CELL %s USING MWXDb_Open_Cell ... \n",progDescrString,
    myCellName,myCellAccessModeChr);

rc = MWXDb_Open_Cell(myCellName, myCellAccessModeChr, &myCellId);

switch(rc) {
    case MWXSuccess:
        MWX_Printf("%s SUCCESSFUL : CELL OPEN.\n", progDescrString);
        break;
    case MWXCellAlreadyOpened:
        MWX_Printf("%s *** WARNING - CELL ALREADY OPENED\n",progDescrString);
        break;
    case MWXNoLibraryOpened:
        MWX_Printf("%s *** ERROR - NO LIBRARY OPENED\n",progDescrString);
        return(MWX_FALSE_SCM);
    case MWXFail:
        MWX_Printf("%s *** ERROR - Cell Open Failure\n",progDescrString);
        return(MWX_FALSE_SCM);
    default:
        MWX_Printf("%s *** UNKNOWN ERROR %d from
        MWXDb_Open_Cell\n",progDescrString,rc);
        return(MWX_FALSE_SCM);
}

/* Now try to open the cell again */

rc = MWXDb_Open_Cell(myCellName, myCellAccessModeChr, &myCellId);
switch (rc) {
    case MWXSuccess:
        MWX_Printf("%s SUCCESSFUL : CELL OPEN.\n", progDescrString);
        break;
    case MWXCellAlreadyOpened:
        MWX_Printf("%s *** WARNING - CELL ALREADY OPENED\n",progDescrString);
        break;
    case MWXNoLibraryOpened:
        MWX_Printf("%s *** ERROR - NO LIBRARY OPENED\n",progDescrString);
        return(MWX_FALSE_SCM);
    case MWXFail:
        MWX_Printf("%s *** ERROR - Cell Open Failure\n",progDescrString);
        return(MWX_FALSE_SCM);
    default:

```

```

        MWX_Printf("%s *** UNKNOWN ERROR %d from
        MWXDb_Open_Cell\n",progDescrString, rc);
    return(MWX_FALSE_SCM);
}
MWX_Printf("%s --- OPENED CELL ID is %d\n", progDescrString, myCellId);

/*-----
    CLOSE THE CELL
    -----*/
MWX_Printf("%s Close Cell and NoSave ...\n",progDescrString);

rc = MWXDb_Close_Cell_And_NoSave(myCellId);

switch(rc) {
    case MWXSuccess:
        MWX_Printf("%s SUCCESSFUL : CELL CLOSE.\n", progDescrString);
        break;
    case MWXNoLibraryOpened:
        MWX_Printf("%s *** ERROR - NO LIBRARY OPENED\n",progDescrString);
        return(MWX_FALSE_SCM);
    case MWXFail:
        MWX_Printf("%s *** ERROR - Cell Close Failure\n",progDescrString);
        return(MWX_FALSE_SCM);
    default:
        MWX_Printf("%s *** UNKNOWN ERROR %d from
        MWXDb_Close_Cell_And_NoSave\n",progDescrString, rc);
        return(MWX_FALSE_SCM);
}

/*-----
    CLOSE THE LIBRARY
    -----*/
MWX_Printf("%s Close Library ...\n",progDescrString);

rc = MWXDb_Close_Lib();

switch(rc) {
    case MWXSuccess:
        MWX_Printf("%s SUCCESSFUL : LIBRARY CLOSE.\n", progDescrString);
        break;
    case MWXNoLibraryOpened:
        MWX_Printf("%s *** ERROR - NO LIBRARY OPENED\n",progDescrString);
        return(MWX_FALSE_SCM);
    case MWXFail:
        MWX_Printf("%s *** ERROR - Library Close Failure\n",progDescrString);
        return(MWX_FALSE_SCM);
    default:
        MWX_Printf("%s *** UNKNOWN ERROR %d from
        MWXDb_Close_Lib\n",progDescrString, rc);
        return(MWX_FALSE_SCM);
}

/*-----
    DONE
    -----*/

MWX_Printf("%s GOODBYE! \n\n",progDescrString);
return(MWX_TRUE_SCM);

```

```

}
void reg_helloworld(){
    MWXSc_Register_Function("helloworld-api",helloworld);
}

```

Example 1-2 shows how to use the stand-alone version to open a cell.

Example 1-2 Accessing a Cell, Stand-Alone Version

```

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include "mwx.h"
#include "mwx_proto.h"
#include "omwx_proto.h"

int main(int argc, char **argv) {
    char *progDescrString = "Hello World Example -";
    static int caseSensitive = 0;
    char myLibName[256] = "testlib";
    MWXLibId_t myLibId;
    char myCellName[256] = "cellTest";
    char myCellAccessModeChr = 'w';
    MWXCellId_t myCellId;

    int rc; /*returned Code */

    /*-----
    Getting Started with the Milkyway Stand-alone C-API. This program opens a
    library, opens a cell, closes the cell, and closes the library.
    -----*/
    MWX_Printf("\n%s HELLO...\n", progDescrString);

    /*-----
    Initialize Open Milkyway
    -----*/
    MWX_Printf("%s Calling MWXDb_omwxInit to INITIALIZE MILKYWAY ...\n",
    progDescrString);

    rc = MWXDb_omwxInit(argc, argv, caseSensitive); /* argc argv used in lmCO */

    switch(rc) {
        case MWXSuccess:
            MWX_Printf("%s SUCCESSFUL : INITIALIZATION ... \n", progDescrString);
            break;
        default:
            MWX_Printf("%s *** ERROR - BAD INITIALIZATION.\n",progDescrString);
            exit(MWXFail);
    }

    /*-----
    OPEN LIBRARY
    -----*/
    MWX_Printf("%s Opening LIBRARY %s USING MWXDb_Open_Lib
    ...\n",progDescrString, myLibName);

    rc = MWXDb_Open_Lib(myLibName,&myLibId);

    switch(rc) {

```

```

        case MWXSuccess:
            MWX_Printf("%s SUCCESSFUL : LIBRARY OPEN.\n", progDescrString);
            break;
        case MWXFail:
            MWX_Printf("%s *** ERROR - Library Open Failure\n",progDescrString);
            exit(MWXFail);
        case MWXNoLicense:
            MWX_Printf("%s *** ERROR - NO LICENSE\n", progDescrString);
            exit(MWXFail);
        default:
            MWX_Printf("%s *** UNKNOWN ERROR %d from
            MWXDb_Open_Lib\n",progDescrString, rc);
            exit(MWXFail);
    }
    MWX_Printf("%s --- OPENED LIBRARY ID is %d\n",progDescrString,myLibId);

/*-----
    OPEN CELL
-----*/
    MWX_Printf("%s Opening CELL %s USING MWXDb_Open_Cell ...\n",progDescrString,
    myCellName,myCellAccessModeChr);

    rc = MWXDb_Open_Cell(myCellName, myCellAccessModeChr, &myCellId);

    switch(rc) {
        case MWXSuccess:
            MWX_Printf("%s SUCCESSFUL : CELL OPEN.\n", progDescrString);
            break;
        case MWXCellAlreadyOpened:
            MWX_Printf("%s *** WARNING - CELL ALREADY OPENED\n",progDescrString);
            break;
        case MWXNoLibraryOpened:
            MWX_Printf("%s *** ERROR - NO LIBRARY OPENED\n",progDescrString);
            exit(MWXFail);
        case MWXFail:
            MWX_Printf("%s *** ERROR - Cell Open Failure\n",progDescrString);
            exit(MWXFail);
        default:
            MWX_Printf("%s *** UNKNOWN ERROR %d from
            WXDb_Open_Cell\n",progDescrString, rc);
            exit(MWXFail);
    }

    MWX_Printf("%s GOODBYE! \n\n",progDescrString);
    return(1);
}

```

[Example 1-3](#) shows the output from running the example programs.

Example 1-3 Runtime Output

```
$ ./OpenMW-1

OpenMW Example 1 - HELLO...
OpenMW Example 1 - Calling MWXDb_omwxInit to INITIALIZE
MILKYWAY...
Milkyway Database Access - Copyright 2002 Synopsys Inc.
PRIVATE Internal Use Only 3.5D
OpenMW Example 1 - SUCCESSFUL: INITIALIZATION ...
OpenMW Example 1 - Opening LIBRARY testlib USING
MWXDb_Open_Lib...
OpenMW Example 1 - SUCCESSFUL: LIBRARY OPEN.
OpenMW Example 1 - --- OPEN LIBRARY ID is 1
OpenMW Example 1 - Opening CELL testcell FOR 'w' USING
MWXDb_Open_Cell...
OpenMW Example 1 - SUCCESSFUL: CELL OPEN.
OpenMW Example 1 - - WARNING - CELL ALREADY OPEN
OpenMW Example 1 - --- OPEN CELL ID is 3
OpenMW Example 1 - Close Cell And NoSave ...
OpenMW Example 1 - SUCCESSFUL: CELL CLOSE (NoSave).
OpenMW Example 1 - GOODBYE!
```

Query Hierarchical Object Example

[Example 1-4](#) demonstrates how to use C-API commands to query hierarchical objects in the design. This example uses the `hierCellInst`, `hierCellInstMaster`, `hierNet`, `hierPort`, `hierPortInstMaster`, and `hierPortVectorInfo` commands.

Example 1-4 Query Hierarchical Object

```
#include<stdio.h>
#include<stdlib.h>
#include "mwx.h"
#include "mda_proto.h"
#include "mwx_proto.h"
#include "omwx_proto.h"

int printHierPortVectorInfo (
    MWXCellId_t cellId,
    MWXObjId_t hierPortVectorInfoId
)
{
    int32 useLeftBound, useRightBound, decLeftBound, decRightBound;

    if (MWXDb_Get_HierPortVectorInfo_useLeftBound(cellId,hierPortVectorInfoId,
                                                &useLeftBound)
        != MWXSuccess) {
        MWX_Printf((char *)"Fail to get hierPortVectorInfo - useLeftBound \n");
        return 0;
    }
    if (MWXDb_Get_HierPortVectorInfo_useRightBound(cellId,hierPortVectorInfoId,
                                                &useRightBound) != MWXSuccess) {
        MWX_Printf((char *)"Fail to get useRightBound for hierPortVectorInfo\n");
        return 0;
    }
}
```



```

    }
    if (MWXDb_Get_HierPortVectorInfo_decLeftBound(cellId,hierPortVectorInfoId,
        &decLeftBound) != MWXSuccess) {
        MWX_Printf((char *)"Fail to get decLeftBound for hierPortVectorInfo\n");
        return 0;
    }
    if (MWXDb_Get_HierPortVectorInfo_decRightBound(cellId,hierPortVectorInfoId,
        &decRightBound) != MWXSuccess) {
        MWX_Printf((char *)"Fail to get decRightBound for hierPortVectorInfo\n");
        return 0;
    }
    MWX_Printf((char*)"\\t\\t\\t- useLeftBound   : %d\\n",useLeftBound);
    MWX_Printf((char*)"\\t\\t\\t- useRightBound  : %d\\n",useRightBound);
    MWX_Printf((char*)"\\t\\t\\t- decLeftBound   : %d\\n",decLeftBound);
    MWX_Printf((char*)"\\t\\t\\t- decRightBound  : %d\\n",decRightBound);
    return 1;
}

int printHierNet (
    MWXCellId_t cellId,
    MWXObjId_t hierNetId
)
{
    char *netName;
    uint32 numberOfHierPortInst;

    if (MWXDb_Get_HierNet_name(cellId,hierNetId,&netName) != MWXSuccess) {
        MWX_Printf((char *)"Fail to get hierNet name \\n");
        return 0;
    }
    if (MWXDb_Get_HierNet_numberOfHierPortInst(cellId,hierNetId,
        &numberOfHierPortInst) != MWXSuccess) {
        MWX_Printf((char *)"Fail to get numberOfPortInst in hierNet\\n");
        return 0;
    }
    MWX_Printf((char*)"\\t\\t\\t- hierNetName           : %s\\n",netName);
    MWX_Printf((char*)"\\t\\t\\t- numberOfHierPortInst :
%d\\n",numberOfHierPortInst);
    return 1;
}

int printHierPortInstMasterInfo (
    MWXCellId_t cellId,
    MWXObjId_t hierPortInstMasterId
)
{
    char *name;
    int32 direction;
    char *hierPortInstMasterName;
    MWXObjId_t hierPortVectorInfoId;

    if (MWXDb_Get_HierPortInstMaster_name(cellId, hierPortInstMasterId, &name)
        != MWXSuccess) {
        MWX_Printf((char *)"Fail to get hierPortInstMasterName\\n");
        return 0;
    }
    if (MWXDb_Get_HierPortInstMaster_direction(cellId,hierPortInstMasterId,
        &direction) != MWXSuccess) {

```

```

        MWX_Printf((char *) "Fail to get direction for hierPortInstMaster\n");
        return 0;
    }
    if (MWXDb_Get_HierPortInstMaster_hierPortVectorInfoId(cellId,
        hierPortInstMasterId, &hierPortVectorInfoId) != MWXSuccess) {
        MWX_Printf((char *) "Fail to get hierPortVectorInfoId\n");
        return 0;
    }
    MWX_Printf((char *) "\t\t- hierPortInstMasterName : %s\n", name);
    MWX_Printf((char *) "\t\t- Direction : %d\n", direction);
    MWX_Printf((char *) "\t\t- hierPortVectorInfoId : %d\n", hierPortVectorInfoId);
    if (hierPortVectorInfoId != 0) {
        if (! printHierPortVectorInfo(cellId, hierPortVectorInfoId) ) {
            return 0;
        }
    }
    }
    return 1;
}

int printHierPortInstInfo (
    MWXCellId_t cellId,
    MWXObjId_t hierPortInstId
)
{
    MWXObjId_t hierPortInstMasterId;
    MWXObjId_t hierNetId1, hierNetId2;

    if (MWXDb_Get_HierPortInst_hierPortInstMasterId (cellId, hierPortInstId,
        &hierPortInstMasterId) != MWXSuccess) {
        MWX_Printf((char *) "Fail to get hierPortInstMastId of hierPortInstId %d\n",
            hierPortInstId);
        return 0;
    }
    MWX_Printf((char *) "\t- hierPortInst Info :\n");
    if (! printHierPortInstMasterInfo(cellId, hierPortInstMasterId) ) {
        return 0;
    }
    if (MWXDb_Get_HierPortInst_hierNet(cellId, hierPortInstId,
    &hierNetId1, &hierNetId2)
        != MWXSuccess) {
        MWX_Printf((char *) "Fail to get hierNetId\n");
        return 0;
    }
    MWX_Printf((char *) "\t\t- hierNetId1 : %d\n", hierNetId1);
    if (hierNetId1) {
        if (! printHierNet(cellId, hierNetId1)) {
            return 0;
        }
    }
    MWX_Printf((char *) "\t\t- hierNetId2 : %d\n", hierNetId2);
    if (hierNetId2) {
        if (! printHierNet(cellId, hierNetId2)) {
            return 0;
        }
    }
    return 1;
}

```

```

int printHierCellInstInfo (
    MWXCellId_t cellId,
    MWXObjId_t hierCellInstId
) {
    int32 rc;
    char *hierCellInstName, *hierCIMName;
    MWXObjId_t hierCIMId, hierPortInstId;
    MWXIteratorId_t hierPortInstIterId;

    MWX_Printf((char *) "Info for HierCellInstId %\n", hierCellInstId);
    if (MWXDb_Get_HierCellInst_name(cellId, hierCellInstId, &hierCellInstName)
        != MWXSuccess) {
        MWX_Printf((char *) "Fail to get hierCellInst name\n");
        return MWXFail;
    }
    if (MWXDb_Get_HierCellInst_hierCellInstMasterId(cellId, hierCellInstId,
        &hierCIMId) != MWXSuccess) {
        MWX_Printf((char *) "Fail to get hierCellInstMasterId\n");
        return MWXFail;
    }
    if (hierCIMId != 0) {
        if (MWXDb_Get_HierCellInstMaster_name(cellId, hierCIMId, &hierCIMName) !=
            MWXSuccess) {
            MWX_Printf((char *) "Fail to get hierCellInstMaster name\n");
            return 0;
        }
        MWX_Printf((char *) "HierCellInstName : %s\n", hierCellInstName);
        MWX_Printf((char *) "\t- HierCellInstMasterName : %s\n", hierCIMName);
        if (MWXDb_Iter_HierCellInstToHierPortInst(cellId, hierCellInstId,
            &hierPortInstIterId) != MWXSuccess) {
            MWX_Printf((char *) "Fail to start iterate hierInst to hierPortInst\n");
            return 0;
        }
        while((rc = MWXDb_Iter_Next(hierPortInstIterId, &hierPortInstId))
            == MWXSuccess) {
            if ( !printHierPortInstInfo(cellId, hierPortInstId) ) {
                MWXDb_Iter_Quit(hierPortInstIterId);
                return 0;
            }
        }
        if (rc == MWXIterDone) {
            MWXDb_Iter_Quit(hierPortInstIterId);
        }
        else {
            MWX_Printf((char *) "Fail to query hierPortInst from hierCellInst\n");
            return 0;
        }
    }
    return 1;
}

MWXSCM_t mwGetHierCellInstInfo (MWXSCM_t list)
{
    char progDescrString[50] = "Get HierCellInst Info -";
    MWXCellId_t cellId;
    MWXObjId_t topHierCellInstId, hierCellInstId;
    MWXIteratorId_t childHierCellInstIterId, hierPortInstIterId;
    static int caseSensitive = 0;

```

```

int rc;

/*-----
GET THE SCHEME LIST
-----*/
if (MWXSc_Scanf (list, "o",&cellId) != 1) {
    MWX_Printf((char*)"Error.Synopsis : getHierCellInstInfo cellId\n");
    return (MWX_FALSE_SCM);
}

/*-----
HELLO
-----*/
MWX_Printf((char*)" \n%s Testing...\n", progDescrString);

if (MWXDb_Get_TopHierCellInst (cellId, &topHierCellInstId) != MWXSuccess) {
    MWX_Printf((char*)"Fail to get top hiercellInstId\n");
    return(MWX_FALSE_SCM);
}
if (!printHierCellInstInfo(cellId,topHierCellInstId) ) {
    MWX_Printf((char*)"Fail to get HierCellInst info \n");
    return (MWX_FALSE_SCM);
}
if (MWXDb_Iter_HierCellInstToChildHierCellInst(cellId, topHierCellInstId,
&childHierCellInstIterId) != MWXSuccess) {
    MWX_Printf((char*)"Fail to iterate childHierCellInst from hierCellInst\n");
}
while ((rc = MWXDb_Iter_Next(childHierCellInstIterId,&hierCellInstId)) ==
MWXSuccess) {
    if (!printHierCellInstInfo(cellId,hierCellInstId) ) {
        MWX_Printf((char*)"Fail to get HierCellInst info \n");
        return (MWX_FALSE_SCM);
    }
}
if (rc == MWXFail) {
    MWX_Printf("Fail to iterate hierCellInst\n");
    return (MWX_FALSE_SCM);
}
if (rc == MWXIterDone) {
    MWXDb_Iter_Quit(childHierCellInstIterId);
}

/*-----
DONE
-----*/
MWX_Printf((char*)" %s DONE TESTING! \n\n",progDescrString);
return (MWX_TRUE_SCM);
}

void reg_getHierCellInstInfo(){

MWXSc_Register_Function((char*)"getHierCellInstInfo",mwGetHierCellInstInfo);
}

```

Create Path Dynamic C-API Example

The following example shows a program that uses the Dynamic C-API to create a path.

```
MWXSCM_t mwCreatePath(MWXSCM_t list) {

    MWXCellId_t cellId;
    int32 rc;
    MWXSCM_t rtn;
    MWXObjId_t pathId;
    char layerName[256];
    int32 dataType, pathType, routeType;
    uint32 width, numOfPoints;
    MWXPoint_t *pointArray = NULL;
    MWXSCM_t pointList;
    MWXSCM_t point;
    MWXSCM_t l;
    double x, y;

    if (MWXSc_Scanf(list, "osddddl", &cellId, layerName, &dataType, &pathType,
        &routeType, &width, &pointList) != 7) {
        MWX_Printf("ERROR -- mwCreatePath requires 7 arguments.\n");
        return (MWX_FALSE_SCM);
    }

    if (! MWXSc_Is_ListSCM(pointList)) {
        MWX_Printf("ERROR -- improper point list.\n");
        return (MWX_FALSE_SCM);
    }

    numOfPoints = 0;
    for (l = pointList; ! MWXSc_Is_NullSCM(l); l = MWXSc_Cdr(l)) {

        /* Get the next coordinate pair.  Make sure its a list. */
        point = MWXSc_Car(l);
        if (! MWXSc_Is_ListSCM(point)) {
            MWX_Printf("ERROR -- improper point.\n");
            free (pointArray);
            return (MWX_FALSE_SCM);
        }

        /* Get the two coordinates. */
        if ((MWXSc_Get_DoubleSCM(MWXSc_Car(point), &x) == MWXFail) ||
            (MWXSc_Get_DoubleSCM(MWXSc_Car(MWXSc_Cdr(point)), &y) == MWXFail)) {
            MWX_Printf("ERROR -- bad point found at position %d.\n", numOfPoints);
            free (pointArray);
            return (MWX_FALSE_SCM);
        }

        /* Add the coordinates to the point array. */
        numOfPoints++;
        pointArray = (MWXPoint_t*) realloc (pointArray,
            sizeof(MWXPoint_t) * numOfPoints);
        pointArray[numOfPoints-1].x = (int32) x;
```

```

        pointArray[numOfPoints-1].y = (int32) y;
    }

    MWX_Printf("INFO -- MWXDb_Create_Path.\n");
    rc = MWXDb_Create_Path (cellId, layerName, dataType, pathType, routeType,
                           width, numOfPoints, pointArray, &pathId);

    switch (rc) {
    case MWXSuccess:
        MWX_Printf("Successfully created path. Path Id = %d.\n", pathId);
        rtn = MWXSc_Make_DBIIdSCM (pathId);
        break;
    case MWXNoLibraryOpened:
        MWX_Printf("ERROR -- No library opened. MWXDb_Create_Path failed.\n");
        rtn = MWX_FALSE_SCM;
        break;
    case MWXFail:
        MWX_Printf("ERROR -- Unable to create path. MWXDb_Create_Path failed.\n");
        rtn = MWX_FALSE_SCM;
        break;
    default:
        MWX_Printf("UNKNOWN ERROR %d from MWXDb_Create_Path.\n", rc);
        rtn = MWX_FALSE_SCM;
    }

    free (pointArray);
    return (rtn);
}

```

Creating and Registering a Function

This section describes the general process for creating, registering, converting, and loading C program functions.

Note:

A license for the dynamic version of the Milkyway C-API is required to register and load a C program function.

The section includes the following topics:

- [Sample Open Cell Function](#)
- [Interpreting the Sample Program Subroutines](#)
- [Registering a Function](#)
- [Loading Modules](#)

Sample Open Cell Function

[Example 1-5](#) shows a C program that uses the Milkyway C-API to reproduce the functionality of the Scheme `dbOpenCell` command using the `MWXDb_Open_Cell` C-API function. All Milkyway C-API functions are structured the same way.

Example 1-5 Opening a Cell

```
#include "mwx.h"
#include "mwx_proto.h"

#ifdef __cplusplus
extern "C" {
    MWXSCM_t myOpenCell();
    void mwInitOpenCell();
}
#endif

MWXSCM_t myOpenCell (MWXSCM_t list)
{
    char cellName[256];
    char mode[256];
    int rc;
    MWXCellId_t cellId;

    if (MWXSc_Scanf(list, "ss", cellName, mode) != 2 ) {
        MWX_Printf("ERROR -- myOpenCell requires 2
arguments\n");
    }

    rc = MWXDb_Open_Cell (cellName, mode[0], &cellId);
    switch (rc) {
        case MWXSuccess:
            return (MWXSc_Make_DBIIdSCM (cellId));
        case MWXFail:
            MWX_Printf("ERROR -- attempt to open cell '%s'
failed\n", cellName);
            return (MWX_FALSE_SCM);
        case MWXCellAlreadyOpened:
            MWX_Printf("ERROR -- cell '%s' is already open\n",
cellName);
            return (MWX_FALSE_SCM);
        case MWXNoLibraryOpened:
            MWX_Printf("ERROR -- no library is open\n");
            return (MWX_FALSE_SCM);
        default:
            MWX_Printf("ERROR -- unknown problem while opening
'%s'\n", cellName);
            return (MWX_FALSE_SCM);
    }
}
```

Interpreting the Sample Program Subroutines

This section describes the task of each code block in [Example 1-5](#).

- `mwX.h` and `mwX_proto.h` Header Files

The declarations that use the API reside within the C header files, which are located at the top of the example. You use these files to declare all of the API

- Data types
- Return codes
- Function prototypes

Include these header files when using any operations defined in the API.

- `extern C` block

If you are using C++ or are using C or C++ on the HP platform, you need to include the `extern "C"` code block and the prototypes for your program to work correctly as shown in the following example:

```
#ifdef __cplusplus
extern "C" {
#include "mwX_proto.h"
int mwOpenCell (MWXCellId_t cellId);
}
#endif
```

The `extern C` code block contains function prototypes of all procedures that must be visible to Milkyway. In the above example, the `extern C` block is enclosed by an `#ifdef` to ensure that the block is used only by C++ compilers.

- `MWXSCM` function

Any procedure that is going to be called from the Milkyway Scheme interface must return the type `MWXSCM_t` and accept parameters of the same type. In the previous sample C program, only a single parameter is received. This does not mean that only a single parameter value was sent. For example,

```
MWXSCM_t myOpenCell (MWXSCM_t list)
{
    char cellName[256];
    char mode[256];
    int rc;
    MWXCellId_t cellId;
```

- `MWXSc_Scanf` function

The `MWXSc_Scanf` function parameter list can be divided into individual parameters using the `MWXSc_Scanf` function. For example,

```
if (MWXSc_Scanf(list, "ss", cellName, mode) != 2 ) {
MWX_Printf("ERROR -- myOpenCell requires 2 arguments\n");
}
```

In this code block, the Scheme list is separated into two strings that are stored in `cellName` and `mode`. Parameters can include

- Strings
- Integers
- Doubles
- Booleans
- Database identifiers
- Lists

The `MWXSc_Scanf` function, like its C function counterpart, does not allocate memory for scanned data.

- `myOpenCell` function

The `myOpenCell` function makes a call to a database function, specifically `MWXDb_Open_Cell`. For example,

```
rc = MWXDb_Open_Cell (cellName, mode[0], &cellId);
```

This operation opens the cell with the specified name and returns a database identifier for the cell. Like the Scheme command `dbOpenCell`, the `myOpenCell` function returns a database identifier for the cell if the cell is successfully opened. If the return code is anything other than `MWXSuccess`, the `cellId` parameter will be undefined.

Note:

A return code is supplied to indicate the result of the operation. Be sure to examine the return codes before using the output.

- `switch` function

[Example 1-6](#) shows a `switch` statement. For the `MWXSuccess` case, the `cellId` is translated for return to the Scheme interpreter using the `MWXSc_Make_DBIdSCM` function.

Example 1-6 A switch Statement

```
switch (rc) {
    case MWXSuccess:
        return (MWXSc_Make_DBIIdSCM (cellId));
    case MWXFail:
        MWX_Printf("ERROR - attempt to open cell '%s'
            failed\n", cellName);
        return (MWX_FALSE_SCM);
    case MWXCellAlreadyOpened:
        MWX_Printf("ERROR - cell '%s' is already open\n",
            cellName);
        return (MWX_FALSE_SCM);
    case MWXNoLibraryOpened:
        MWX_Printf("ERROR - no library is open\n");
        return (MWX_FALSE_SCM);
    default:
        MWX_Printf("ERROR - unknown problem while opening
            '%s'\n", cellName);
        return (MWX_FALSE_SCM);
}
```

If the operation fails, `MWX_FALSE_SCM` is returned, which is equivalent to `#f` in the Scheme interface.

Note:

The Scheme interpreter does not directly understand the same data types as the C programming language does. Therefore, values exchanged between C and Scheme must be translated. There are several C-API functions for performing the translation, such as `MWXSc_Get_DBIIdSCM` and `MWXSc_Make_DBIIdSCM`.

(For the list of functions, see [Chapter 7, “Scheme Interpreter Functions.”](#))

In [Example 1-5](#) the translation of the input parameters takes place inside the `MWXSc_Scanf` procedure, so no additional steps are necessary:

Registering a Function

If you are working with the dynamic version of the Milkyway C-API and have already created a C program function similar to the program described in the previous section, you next need to register it and load it into the Milkyway environment. The following procedure shows how to register and load a C program function.

Procedure for Registering and Loading a New Function

This procedure describes how to register, convert, and load a C program function.

1. Create the C program that you want to use with the products related to Astro (see [Sample Open Cell Function](#)).

2. After you have created all of the constructs for your C program, register your function command by calling the `MWXSc_Register_Function` C-API function in your C code.

Milkyway will register the C function `myOpenCell` with the Scheme command alias `my_Open_Cell` as shown below:

```
void registerMyCommands()
{
    MWXSc_Register_Function ("my_Open_Cell", myOpenCell);
}
```

You must call the registration function once for every C program function that you want to call from Milkyway. For instance, executing the `my_Open_Cell` Scheme command calls your `myOpenCell` C function.

3. Close and save your files, then compile them to object code.
4. After you have your object files, convert them into the form expected by Milkyway, using the `AvantAr` command. Milkyway can then dynamically load these converted files.

In this example, the command will generate a file called `OpenCell.aa`:

```
AvantAr OpenCell.o
```

5. Load an object file with the `loadModule` Scheme command. Provide the name of a C function to call as an argument to the `loadModule` command. For example,

```
(loadModule "registerMyCommands" '("OpenCell.aa"))
(define cellId (my_Open_Cell "killUnit" "r"))
(if (not (equal? cellId #f))
    .
    .
    .)
```

As a result, the `OpenCell.aa` file is dynamically bound to Milkyway, which calls the `registerMyCommands` function and registers the `myOpenCell` function under the Scheme counterpart, `my_Open_Cell`. When the `load` command completes successfully, the command will return

```
#UNSPECIFIED (not #f, not#f)
```

Note:

The word `UNSPECIFIED` will not be printed out in the command window.

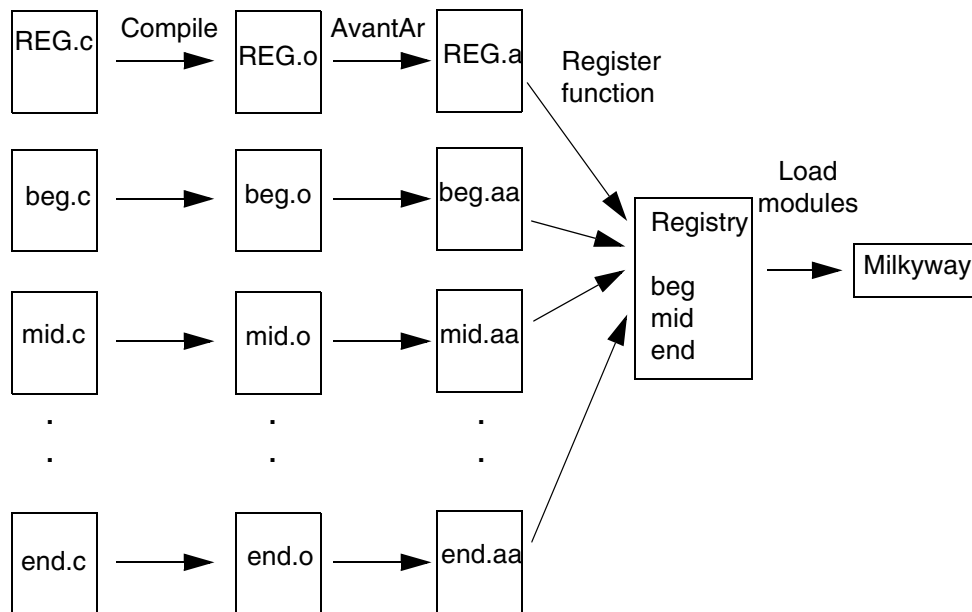
6. You can now call the `my_Open_Cell` command from Scheme.

Note:

An object file cannot be reloaded. If a C file is changed and recompiled, exit Milkyway and restart it before loading the object file again.

Figure 1-6 illustrates the procedure.

Figure 1-6 Function Creation Process



Loading Modules

The Scheme `loadModule` command dynamically links object files to a Milkyway, Astro, or JupiterXT executable. And, because the user modules can interact, one module can call functions defined in another module.

The Sun and HP dynamic linkers have slightly different behaviors that you must consider when loading multiple, interacting object files.

- The Sun linker requires that you define symbols before using them, so the linking order is relevant. For example, if module A defines a function used in module B, you must load module A into Milkyway before you load module B.
- The HP linker is a little more robust. Rather than requiring that you define symbols before you can use them, it only requires that you define symbols before you call them. This means that when user-defined modules use symbols defined in other modules, the linker checks for the existence of these symbols only when they are called during execution, not when they are encountered at link time. For example, if module A defines a function used in module B, it does not matter which module is loaded first, as long as module A has been loaded before the function it contains is called.

When writing your code, you should rely on a more restrictive approach, such as the Sun approach, to maintain platform independence on the two machines.

C-API Function Return Codes

All functions return a result code such as `MWXSuccess` or `MWXFailure`.

Caution!

Always check the return code after every C-API function call.

[Table 1-1](#), [Table 1-2](#), and [Table 1-3](#) list the codes returned by the Milkyway C-API functions.

[Table 1-1](#) lists the basic codes.

Table 1-1 Basic Codes

Value	Return Code Identifier	Description
0	<code>MWXSuccess</code>	Function completed successfully. If a string or array is returned then memory was allocated for it.
1	<code>MWXFail</code>	Function did not complete successfully.
2	<code>MWXNoLicense</code>	The appropriate runtime license was not found.

[Table 1-2](#) lists the library and cell codes.

Table 1-2 Library and Cell Codes

Value	Return Code Identifier	Description
201	<code>MWXNoLibraryOpened</code>	Library unable to be opened
202	<code>MWXLibraryAlreadyOpened</code>	Library is already open. Not an error.
203	<code>MWXCellAlreadyOpened</code>	Cell is already open. Not an error.
204	<code>MWXNoCellOpened</code>	Cell unable to be opened

Table 1-3 lists the cell object codes.

Table 1-3 Cell Object Codes

Value	Return Code Identifier	Description
301	MWXInvalidCellIdOrObjId	Invalid cell or object ID specified.
302	MWXInvalidCellId	Invalid cell ID specified.
303	MWXInvalidObjId	Invalid object ID specified.
304	MWXObjTypeMismatch	Object type mismatch.
305	MWXInvalidStringId	Invalid string ID specified.
306	MWXNullStringArgument	The string argument is NULL.
307	MWXSetDBStringFail	Setting DB string failed.
308	MWXNullPointArgument	The point argument is NULL.
309	MWXInvalidDataFieldName	Invalid data field name specified.
310	MWXInvalidInternalFileId	The internal file ID is invalid.
311	MWXGetAttributeEntryFail	
312	MWXGetObjTypeNameFail	Could not get object type name.
313	MWXGetDBStringIdFail	Could not get DB string ID.
314	MWXInvalidArgumentValue	The argument value is invalid.
315	MWXInvalidArgumentElementValue	The argument element value is invalid.
316	MWXNullArgumentValue	The argument value is NULL.
317	MWXNumberOfPointsMismatch	The number of points are mismatched.
318	MWXUnpackPointsFail	
319	MWXTooManyUnpackedPoints	Too many unpacked points are specified.

Table 1-3 Cell Object Codes (Continued)

Value	Return Code Identifier	Description
320	MWXNullStringValue	The string value is NULL.

2

Working With Milkyway Database Objects

This chapter describes general mechanisms for working with Milkyway database objects. It contains the following sections:

- [Working With Libraries](#)
- [Working With Cells](#)

Working With Libraries

A Milkyway 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. But instead of copying cells to multiple libraries, Milkyway references the other libraries, which, in such a case, are called *reference libraries*.

A Milkyway library is implemented as a UNIX directory containing the following:

- Library subdirectories containing each cell type in the library
- A library information file (lib), which is a binary file containing
 - information compiled from a technology file
 - a cell library format (CLF) file
 - the cell files inside the library directory

You cannot directly edit the library information file. However, you can change the technology and CLF files and then use Milkyway to unload and reload these files. When uploading one of these files, Milkyway presents the information in an editable format that you can modify using a UNIX text editor.

Sample libraries are available in the directory where you installed Milkyway.

Naming Libraries

Guidelines for naming libraries include the following:

- The first character should be an alphabetic character.
- The remaining characters can be any combination of the following:
 - Any alphanumeric character
 - An underscore (_)
- The name length should not exceed 127 characters.

Accessing Libraries

You can open a library in either of two modes:

- Read-only mode, which lets you open the library but not make any changes to it
- Write mode, which lets you open the library and make changes

Note:

Milkyway can have only one library open in write mode.

When you open a cell library, Milkyway assigns a database object ID to it. This ID is valid only until the library is closed or the Milkyway session ends. Some commands need this library ID as an argument. Because you can open only one library at a time in write mode, the `MWXDb_Open_Lib` function specifies a library ID. Most other Milkyway C-API functions assume this global library ID. For more information, see [“MWXDb_Open_Lib” on page 4-258](#).

Specifying A Library ID

To specify a library ID, provide a variable for the library ID when you open a cell library, as shown in [Example 2-1](#).

Example 2-1 Specifying a Variable ID

```
MWXLlibId_t myLibId;
int rc; /* return code */

rc = MWXDb_Open_Lib("test", &myLibId);

switch (rc) {
    case MWXSuccess:
        MWX_Printf("SUCCESSFUL: LIBRARY OPEN.\n");
        break;
    case MWXFail:
        MWX_Printf("*** ERROR - Library Open Failure\n");
        exit(MWXFail);
    case MWXNoLicense:
        MWX_Printf("*** ERROR - NO LICENSE\n");
        exit(MWXFail);
    default:
        MWX_Printf("*** UNKNOWN ERROR %d from MWXDb_Open_Lib\n", rc);
        exit(MWXFail);
}
MWX_Printf("--- OPEN LIBRARY ID is %d\n", myLibId);
```

Interpreting the Variable ID Example

In [Example 2-1](#),

- The first two lines declare the variables for the library ID (`myLibId`) and a return code
- The third line opens the library and creates a top-level definition (in `myLibId`) for the database object ID of the library

Recovering a Variable ID

If the library ID is not bound to a variable or the variable is lost, you recover it using one of the following techniques:

- Use [MWXDb_Get_CurrentLibId](#):

```
MWXLlibId_t myLibId;
int rc;          /* return code */

rc = MWXDb_Get_CurrentLibId(&myLibId);
```

- If a cell is already open in your library, you can use [MWXDb_Get_LibId_ByCellId](#).

```
MWXCellId_t myCellId;
MWXLlibId_t myLibId;

int rc;          /* return code */

rc = MWXDb_Get_LibId_ByCellId(myCellId, &myLibId);
```

Functions Used on Libraries

[Table 2-1](#) lists Milkyway C-API functions for working with library objects.

Table 2-1 Functions for Working With Library Objects

Function purpose	Milkyway C-API function
Create library	MWXDb_Create_Lib
Open library	MWXDb_Open_Lib
Get library ID	MWXDb_Get_CurrentLibId
Copy library	MWXDb_Copy_Lib
Rename library	MWXDb_Rename_Lib
Close and save library	MWXDb_Close_Lib
Reference other library	MWXDb_Add_RefLib
Replace referenced library	MWXDb_Replace_RefLib_ByName
Unlink referenced library	MWXDb_Delete_RefLib_ByName

Table 2-1 Functions for Working With Library Objects (Continued)

Function purpose	Milkyway C-API function
Set or replace technology file for open library	MWXDb_Set_TechFileName MWXDb_Replace_TechInfo
Attach file to a library	MWXDb_Create_LibAttachFile MWXDb_Store_LibAttachFile
Delete attached file from library	MWXDb_Delete_LibAttachFile
Open file attached to library	MWXDb_Open_LibAttachFile MWXDb_Open_RefLibAttachFile

Working With Cells

In Milkyway, a representation of a design is called a cell. But for any design, there are many possible kinds of cells as shown in the following list:

- A layout cell, which represents the physical layout of a design
- A pin/blockage cell, which represents the pins and blockage areas (that is, 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
- A hierarchical netlist cell, which references other netlist cells
- 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

Naming Cells

The guidelines for naming cells are as follows:

- The first character must be alphabetic.
- The remaining characters can be any of the following:

- Any alphabetic character
- Any numeric character
- Underscore (_)
- The name must not exceed 31 characters.

Naming Conventions

Table 2-2 lists the different types of cells and the naming conventions for the cells and their corresponding binary files. Although you can assign any name of up to 32 characters to a cell, the extension (after the “.”) cannot exceed 4 characters.

Table 2-2 Cell Types, Versioning, and Binary File Names

Cell type	First version of cell name	Corresponding binary file name
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
gap fill	<layoutCellName>.GAP;1	GAP/<layoutCellName>:1
hierarchical netlist	<netlistCellName>.HNET;1	HNET/<netlistCellName>:1

Version Numbering

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.

Accessing Cells

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

- Read-only mode, which lets you open the cell but does not let you make any changes
- Write mode, which lets you open the cell and make changes

Specifying a Cell ID

When you open a cell, Milkyway assigns it a cell ID. This ID is valid only for the current session and while the cell is open. Most Milkyway C-API functions 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. The following example shows how to open a cell.

Example 2-2 Specifying a Cell ID

```
...
MWXCellId_t cellId;
int rc;          /* return code */

rc = MWXDb_Open_Cell("toprt", "w", &cellId);
switch (rc) {
    case MWXSuccess:
        MWX_Printf("SUCCESSFUL: CELL OPEN.\n");
        break;
    case MWXCellAlreadyOpened:
        MWX_Printf("*** WARNING - CELL ALREADY OPEN\n");
        break;
    case MWXNoLibraryOpened:
        MWX_Printf("*** ERROR - NO LIBRARY OPENED\n");
        exit(MWXFail);
    case MWXFail:
        MWX_Printf("*** ERROR - Cell Open Failure\n");
        exit(MWXFail);
    default:
        MWX_Printf("*** UNKNOWN ERROR %d from MWXDb_Open_Cell\n", rc);
        exit(MWXFail);
}

/* code that accesses the cell in cellId */

rc = MWXDb_Close_Cell(cellId);
```

Interpreting the Cell ID Example

In [Example 2-2](#), the [MWXDb_Open_Cell](#) function opens `toprt` in write mode. Milkyway enforces cell access control as follows:

- When you open a cell in read-only mode, Milkyway 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, Milkyway locks the cell from write access by other users by creating a lock file (`<cellName>.lock`). When you close the cell, Milkyway 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:

```
rc = MWXDb_Change_Cell_AccessMode(cellId, "w");
```

The `cellId` argument is the internal ID of any open cell in the current library to which you want to make changes. For more information, see [“MWXDb_Change_Cell_AccessMode” on page 4-15](#).

Specifying a Cell ID

Most of the Milkyway C-API functions require you to specify a cell ID.

When you open the cell by using [MWXDb_Open_Cell](#), you must supply as the third parameter a pointer to the location where the cell ID is returned.

When a static variable is used, as shown in the example on [page 2-7](#), `&cellID` is passed to the function as a pointer to the location of that static variable. Furthermore, `cellID` is used to specify which cell to close when you are using [MWXDb_Close_Cell](#). For all later references to the cell, you use that same variable.

Functions Used on Cells

[Table 2-3](#) lists Milkyway C-API functions for working with cell objects.

Table 2-3 Functions for Working With Cell Objects

Function purpose	Milkyway C-API function
Create cell	MWXDb_Create_Cell
Delete cell	MWXDb_Delete_Cell

Table 2-3 *Functions for Working With Cell Objects (Continued)*

Function purpose	Milkyway C-API function
Copy cell	MWXDb_Copy_Cell MWXDb_Copy_CellFromAnotherLib MWXDb_Copy_CellWithTIM
Rename cell	MWXDb_Rename_Cell
Cell existence check	MWXDb_Check_Cell_Exist
Open existing cell	MWXDb_Open_Cell
Find ID of cell opened for edit	MWXDb_Get_EditCell
Change access mode of cell	MWXDb_Get_Cell_AccessMode MWXDb_Change_Cell_AccessMode
Close and save cell	MWXDb_Close_Cell_And_Save MWXDb_Close_Cell
Close cell without saving	MWXDb_Close_Cell_And_NoSave MWXDb_Abort_Cell
Save an open cell	MWXDb_Save_Cell
Revert to last saved version	MWXDb_RollBack_Cell
Get library ID of cell	MWXDb_Get_LibId_ByCellId
Attach file to cell	MWXDb_Create_CellAttachFile MWXDb_Store_CellAttachFile
Open files attached to cell	MWXDb_Open_CellAttachFile
Delete files attached to cell	MWXDb_Delete_CellAttachFile
Create or modify user-defined cell properties	MWXDb_Create_Cell_Property MWXDb_Delete_CellProperty_ByName MWXDb_Get_AllCellProperties MWXDb_Get_CellProperty_ByName MWXDb_Update_CellProperty_ByName
Modify text string in cell	MWXDb_Replace_CellTextChar MWXDb_Replace_CellTextString MWXDb_Truncate_CellText

Table 2-3 *Functions for Working With Cell Objects (Continued)*

Function purpose	Milkyway C-API function
Modify or update cell hierarchy	MWXDb_Force_UpdateCellHierarchy MWXDb_Replace_CellInst MWXDb_Replace_ChildCell
Mark instances of a cell as standard cell	MWXDb_Set_StdCell
Equate two cells electrically or logically	MWXDb_Create_CellEEQClass MWXDb_Create_CellLEQClass
Get electrical or logical equivalence of two cells	MWXDb_Get_Cell_EEQClass MWXDb_Get_Cell_LEQClass
Unequate two cells electrically or logically	MWXDb_Clear_CellEEQClass MWXDb_Clear_CellLEQClass
Get, create, or set other cell attributes	MWXDb_Get_CellBBox MWXDb_Get_CellBoundary MWXDb_Get_CellType_Bit MWXDb_Create_CellBoundary MWXDb_Create_CellRegion MWXDb_Create_CellRow MWXDb_Set_CellLModelSubType MWXDb_Set_CellLayoutSubType MWXDb_Clear_CellMustJoinPin MWXDb_Set_CellMustJoin2Pin MWXDb_Set_CellMustJoinPinPattern MWXDb_Set_CellType MWXDb_Set_CellType_Bit

3

Database Objects, Object Fields, and Field Accessor Functions

This chapter describes the following Milkyway database components:

- Milkyway database objects
- Object fields
- Field accessor functions

It contains the following sections:

- [Accessing Objects in Cells](#)
- [Object Types and Object Fields](#)
- [Object Field Descriptions and Field Accessor Functions](#)
- [Common Object Fields](#)

Accessing Objects in Cells

The Milkyway database is organized in a hierarchy consisting of *libraries*, *cells*, *objects*, and *fields*. A library contains top-level designs and the cells used in those designs. A cell is a representation of a design. Objects in the Milkyway database are always associated with cells.

To retrieve objects and their fields, you must first open a library and then a cell contained within that library. Once you open a cell, you can retrieve groups of objects with identical object types. Finally, when an object is available, you can use the Milkyway C-API functions to retrieve specific data fields for the object or modify the fields, attributes, or relations of that object. Objects and their field descriptions are described in [“Object Field Descriptions and Field Accessor Functions” on page 3-9](#).

Opening a Library

You use [MWXDb_Open_Lib](#) to open a library for editing. When you open a library, Milkyway assigns a database object ID (library ID) to it. You provide a variable (`myLibId`, for example) for the library ID. For example,

```
MWXLlibId_t myLibId;
int rc; /* return code */

rc = MWXDb_Open_Lib("test", &myLibId);
```

The library ID is valid until the library is closed (see [MWXDb_Close_Lib](#)) or the Milkyway session ends.

Also, you can open a library by using the `MWXDb_Open_Library_ByAccessMode` function. This function can open a library in either read-only or read-write mode.

Note:

Milkyway can have only one library open in write mode.

Opening a Cell

You use [MWXDb_Open_Cell](#) to open a cell. You can open a cell for read or write access. When you open a cell, Milkyway assigns a database object ID (cell ID) to it. The cell ID is valid for the current session and while the cell is open. You provide a variable (`cellId` for example) for the cell ID. For example,

```

MWXCellId_t cellId;
int rc;          /* return code */

rc = MWXDb_Open_Cell("toprt","w", &cellId);

```

Getting Objects From Cells

You can use [MWXDb_Get_Objects_ByType](#) as a general mechanism for retrieving objects from a cell by retrieving all objects of a given type within a specific cell.

Once an object is available, you can then use [MWXDb_Fetch_Object](#) to retrieve all the object fields.

`MWXDb_Get_Objects_ByType` is similar to the Scheme `db-foreach` command. Rather than starting a loop, however, `MWXDb_Get_Objects_ByType` returns an array containing all database identifiers of the objects requested. The following example demonstrates how to retrieve all the ports belonging to a particular cell:

```

MWXCellId_t cellId; /* handle to cell object */
int rc;             /* return code */
int32 objCount;     /* returned number of objects */
MWXObjId_t *objIdArray; /* returned object */
...
rc = MWXDb_Get_Objects_ByType (cellId, /* input */
                               MWX_PORT, /* input */
                               &objCount, /*output*/
                               &objIdArray); /*output*/

```

In the above example, the requested object type is identified by `MWX_PORT`, which (along with constant values for each of the other object types) appears in the `mw.h` header file. All of the returned port database identifiers are placed into `objIdArray` and the number of array entries is returned in `objCount`.

The `objCount` and `objIdArray` can now be used in a loop to cycle through all of the cell ports, just as the `db-foreach` command is used in Scheme. The example below demonstrates how to get the names of each port.

```

for (i = 0; i < objCount; i++)
    rc = MWXDb_Get_Port_name(cellId,*(objIdArray+i), &nstr);
    if (rc != MWXSuccess) {continue;}
    ...
    free(nstr);

```

Note:

In the above example, a string pointer with the name of the port is returned. The `MWXDb_Get_Port_name` function (like all functions that return strings) allocates memory for the string. It is up to the programmer to free this memory when the string is no longer needed. No memory is allocated for string or array outputs of `MWXDb_Get_*` functions if the function returns anything other than `MWXSuccess`.

Getting Data Fields of Objects

You use the general class of `MWXDb_Get_object_field` functions to retrieve a particular data field from a Milkyway database object; where *object* is an object name, and *field* is a field name within *object*. For example, `MWXDb_Get_CellInfo_name` retrieves the cell name of a `Cellinfo` object.

The first two arguments to almost all `MWXDb_Get_object_field` functions are the database identifier of the cell for which data is being retrieved and the database identifier of the object that owns the field being retrieved. These two arguments are inputs.

Note:

The `CellEquivalentClass` object is the only exception. It takes a *library* identifier instead of a *cell* identifier.

The remaining arguments for each `MWXDb_Get_object_field` function can be found in their respective function prototypes. (The prototypes follow the individual object descriptions.) These arguments are outputs and are used to return the database values requested. Because these arguments are outputs, addresses of locations into which to place the results must be passed to the function. The following example demonstrates how to retrieve the points of a `Path` object.

```
uint32 numberOfPoints;
MWXPoint_t *points;
...
MWXDb_Get_Path_points(cellId, pathId, &numberOfPoints,
    &points);
```

The x, y coordinate pairs are returned in the `points` array, and `numberOfPoints` indicates the number of array entries. The addresses of the output parameters are passed to the function, and the memory for the `points` array is allocated by the `MWXDb_Get_Path_points` function.

Note:

Because `MWXDb_Get_Object_field` functions that return strings, arrays, or `MWXPoint_t` allocate memory for these objects, you must recover that memory when the data is no longer needed.

Deleting Objects

You can use the following Milkyway C-API functions to delete cell objects:

- [MWXDb_Delete_Object](#)
- [MWXDb_Delete_ObjectArray](#)
- [MWXDb_Delete_Objects_ByLayer](#)

Object Types and Object Fields

This section lists the object types and common object fields. [Table 3-1](#) list the object types. [Table 3-2](#) lists the common object fields.

Object Types

[Table 3-1](#) shows all the Milkyway objects, including cell and library, and the objects within cells. Each cell object shows a reference to another table that details the fields of that object.

Table 3-1 Object Types

Object Name	Object Type (from mw.h)
BaseArray	MWXDb_Get_BaseArray_*
Boundary	MWXDb_Get_Boundary_*
BPVContactArray	MWXDb_Get_BPVContactArray_*
CEdge	MWXDb_Get_CEdge_*
Cell	MWXCellId_t
CellEquivalentClass	MWXDb_Get_CellEquivalentClass_*
CellInfo	MWXDb_Get_CellInfo_*
CellInst	MWXDb_Get_CellInst_*
CellInstArray	MWXDb_Get_CellInstArray_*
CellInstMaster	MWXDb_Get_CellInstMaster_*
CellRegion	MWXDb_Get_CellRegion_*

Table 3-1 Object Types (Continued)

Object Name	Object Type (from mwx.h)
CellRow	MWXDb_Get_CellRow_*
ClockOptData	MWXDb_Get_ClkOptData_*
CLFRecord	MWXDb_Get_CLFRecord_*
Contact	MWXDb_Get_Contact_*
ContactArray	MWXDb_Get_ContactArray_*
DelayTable	MWXDb_Get_DelayTable_*
ErrorComponent	MWXDb_Get_ErrorComponent_*
ErrorObj	MWXDb_Get_ErrorObj_*
ErrorType	MWXDb_Get_ErrorType_*
ExpCellInst	MWXDb_Get_ExpCellInst_*
ExpPinInst	MWXDb_Get_ExpPinInst_*
ExpPortInst	MWXDb_Get_ExpPortInst_*
Glink	MWXDb_Get_Get_Glink_*
Group	
HorizontalWire	MWXDb_Get_HorizontalWire_*
Library	MWXLlibId_t
Net	MWXDb_Get_Net_*
Path	MWXDb_Get_Path_*
Pin	MWXDb_Get_Pin_*
PinAccessEdge	MWXDb_Get_PinAccessEdge_*
PinInstMaster	MWXDb_Get_PinInstMaster_*
PlanGroup	MWXDb_Get_PlanGroup_*

Table 3-1 Object Types (Continued)

Object Name	Object Type (from mwx.h)
Polygon	MWXDb_Get_Polygon_*
Port	MWXDb_Get_Port_*
PortFunction	MWXDb_Get_PortFunc_*
PortInst	MWXDb_Get_PortInst_*
PortInstMaster	MWXDb_Get_PortInstMaster_*
Port-to-Port Delay	MWXDb_Get_PortToPortDelay_*
Property	MWXObjId_t
Rectangle	MWXDb_Get_Rectangle_*
SchArc	MWXDb_Get_SchArc_*
SchBundle	MWXDb_Get_SchBundle_*
SchCircle	MWXDb_Get_SchCircle_*
SchCustom	MWXDb_Get_SchCustom_*
SchGlobalNet	MWXDb_Get_SchGlobalNet_*
SchJunction	MWXDb_Get_SchJunction_*
SchLines	MWXDb_Get_SchLines_*
SchNameAnnotation	MWXDb_Get_SchNameAnnotation_*
SchPinInst	MWXDb_Get_SchPinInst_*
SchPin	MWXDb_Get_SchPin_*
SchPolygon	MWXDb_Get_SchPolygon_*
SchPort	MWXDb_Get_SchPort_*
SchProperty	MWXDb_Get_SchProperty_*
SchRectangle	MWXDb_Get_SchRectangle_*

Table 3-1 Object Types (Continued)

Object Name	Object Type (from mwx.h)
SchRipper	MWXDb_Get_SchRipper_*
SchSegment	MWXDb_Get_SchSegment_*
SchSubnet	MWXDb_Get_SchSubnet_*
SchText	MWXDb_Get_SchText_*
Text	MWXDb_Get_Text_*
TilePattern	MWXDb_Get_TilePattern_*
VerticalWire	MWXDb_Get_VerticalWire_*
ViaRegion	MWXDb_Get_ViaRegion_*
WireDirection	MWXDb_Get_WireDirection_*
WireMaster	MWXDb_Get_WireMaster_*
WireTrack	MWXDb_Get_WireTrack_*

Common Object Fields

[Table 3-2](#) lists the common object fields and includes links to the object field tables.

Table 3-2 Common Object Fields

Common Object Field
direction
port
accessDirection
cellFlag Bit
recType Bit
routeType Bit

Table 3-2 Common Object Fields

Common Object Field
statusFlag Bit
tableType Bit
transformCode

Object Field Descriptions and Field Accessor Functions

This section describes the Milkyway database objects. For each object, the following information is provided:

- Object field descriptions
- Field accessor function prototypes

Object fields that are common among many of the objects are described in [“Common Object Fields” on page 3-105](#).

BaseArray Object

The field descriptions for the BaseArray object are listed in [Table 3-3](#).

Table 3-3 BaseArray Object Field Descriptions

Field	Output	Description
baseArrayClass	int32	Base array class with values: 0 - default class 1 - core 2 - I/O
name	char*	Name of the tile assigned to the base array (tile definitions appear in the technology file)
bbox	4 of int32	Two x/y-coordinate pairs representing the lower-left and upper-right corners of the bounding box of the array
subTileFlag	int32	Cell subTile flag (1 = default value)

Table 3-3 BaseArray Object Field Descriptions (Continued)

Field	Output	Description
tileWidth	int32	Width of the tile assigned to the base array (tile definitions appear in the technology file)
tileHeight	int32	Height of the tile assigned to the base array (tile definitions appear in the technology file)
numberOfCellRow	int32	Number of cell rows contained in the base array

The prototypes for the BaseArray accessor functions are

```
int MWXDb_Get_BaseArray_baseArrayClass (MWXCellId_t cellId,
    MWXObjId_t objId, uint32 *value);

int MWXDb_Get_BaseArray_name (MWXCellId_t cellId,
    MWXObjId_t objId, char **string);

int MWXDb_Get_BaseArray_bbox (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *left, int32 *bottom, int32 *right, int32 *top);

int MWXDb_Get_BaseArray_subTileFlag (MWXCellId_t cellId,
    MWXObjId_t objId, uint32 *value);

int MWXDb_Get_BaseArray_tileWidth (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_BaseArray_tileHeight (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_BaseArray_numberOfCellRow (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);
```

Additional functions related to BaseArray are

- [MWXDb_Create_BaseArray](#)
- [MWXDb_Create_BaseArrayRow](#)
- [MWXDb_Create_CoreBaseArray](#)
- [MWXDb_Create_Row](#)

Boundary Object

The field descriptions for the Boundary object are listed in [Table 3-4](#).

Table 3-4 Boundary Object Field Descriptions

Field	Output	Description
type	int32	Boundary type
drawable	int32	Drawable condition of the boundary
flag	uint32	Internal field
bbox	4 of int32	Two x/y-coordinate pairs representing the lower-left and upper-right comers of bounding box of the boundary
netId	MWXObjId_t	Database identifier of the net attached to the boundary
layer	int32	The layer number that the boundary belongs to
dataType	int32	The data type of the boundary
numberOfPoint	uint32	Number of points that generate the boundary
points	MWXPoint_t*	Coordinate points that generate the boundary

The prototypes for the Boundary accessor functions are

```
int MWXDb_Get_Boundary_type (MWXCellId_t cellId,
                             MWXObjId_t objId, int32* type);

int MWXDb_Get_Boundary_drawable (MWXCellId_t cellId,
                                 MWXObjId_t objId, int32 *drawable);

int MWXDb_Get_Boundary_flag (MWXCellId_t cellId,
                             MWXObjId_t objId, uint32 *flag);

int MWXDb_Get_Boundary_bbox (MWXCellId_t cellId,
                             MWXObjId_t objId, int32 *left, int32 *bottom, int32 *right, int32 *top);

int MWXDb_Get_Boundary_netId (MWXCellId_t cellId,
                              MWXObjId_t objId, MWXObjId_t*netId);

int MWXDb_Get_Boundary_layer (MWXCellId_t cellId,
```

```

        MWXObjId_t objId, int32*layer);

int MWXDb_Get_Boundary_dataType (MWXCellId_t cellId,
        MWXObjId_t objId, int32*dataType);

int MWXDb_Get_Boundary_numberOfPoints (MWXCellId_t cellId,
        MWXObjId_t objId, int32*numberOfPoints);

int MWXDb_Get_Boundary_points P_ ((MWXCellId_t cellId,
        MWXObjId_t objId, int32 *numOfPoint, MWXPoint_t **points));

```

Additional functions related to Boundary are

- [MWXDb_Create_CellBoundary](#)
- [MWXDb_Get_CellBoundary](#)
- [MWXDb_Get_PrBoundary](#)

BPVContactArray Object

The field description for the BPVContactArray object is listed in [Table 3-5](#).

Table 3-5 BPVContactArray Object Field Description

Field	Output	Description
flag	int32	BPV contact array class with values: 0 - bit map 1 - data type 2 - metal size

The prototype for the BPVContactArray accessor function is

```

int MWXDb_Get_BPVContactArray_flag (MWXCellId_t cellId,
        MWXObjId_t objId, int32 *value);

```

CEdge Object

The field descriptions for the CEdge object are listed in [Table 3-6](#).

Table 3-6 CEdge Field Descriptions

Field	Output	Description
nodeId1	uint32	Node 1 ID

Table 3-6 CEdge Field Descriptions

Field	Output	Description
nodeId2	uint32	Node 2 ID
nomCap	float32	Nominal capacitance

The prototypes for the CEdge accessor functions are

```
int MWXDb_Get_CEdge_nodeId1(MWXCellId_t cellId,
                             MWXObjId_t CEdgeId, uint32 *nodeId1);

int MWXDb_Get_CEdge_nodeId2(MWXCellId_t cellId,
                             MWXObjId_t CEdgeId, uint32 *nodeId2);

int MWXDb_Get_CEdge_nomCap(MWXCellId_t cellId,
                             MWXObjId_t CEdgeId, float *value);

int MWXDb_Set_CEdge_nodeId1(MWXCellId_t cellId,
                             MWXObjId_t objId, uint32 value);

int MWXDb_Set_CEdge_nodeId2(MWXCellId_t cellId,
                             MWXObjId_t objId, uint32 value);

int MWXDb_Set_CEdge_nomCap(MWXCellId_t cellId,
                             MWXObjId_t objId, float value);
```

CellEquivalentClass Object

The field descriptions for the CellEquivalentClass object are listed in [Table 3-7](#).

Table 3-7 CellEquivalentClass Object Field Descriptions

Field	Output	Description
cellName1	char*	Name of the cell that is equivalent to the cell named in cellName2
cellName2	char*	Name of the cell that is equivalent to the cell named in cellName1
classType	int32	One (bit-and with constant dbbCellEEQClass) if the two cells named in cellName1 and cellName2 are electrically equivalent, two (bit-and with constant dbbCellLEQClass) if they are logically equivalent

The prototypes for the CellEquivalent accessor functions are

```

int MWXDb_Get_CellEquivalentClass_cellName1 (MWXLibId_t
    LibId, MWXObjId_t objId, char **string);

int MWXDb_Get_CellEquivalentClass_cellName2 (MWXLibId_t
    LibId, MWXObjId_t objId, char **string);

int MWXDb_Get_CellEquivalentClass_classType (MWXLibId_t
    LibId, MWXObjId_t objId, int32 *value);

```

Note:

The first parameter of CellEquivalentClass functions is a library identifier, not a cell identifier.

CellInfo Object

The field descriptions for the CellInfo object are listed in [Table 3-8](#).

Table 3-8 CellInfo Object Field Descriptions

Field	Output	Description
name	char*	Name of the associated cell.
creationDate	6 of int32	Month, day, year, hour, minute, and second that the associated cell was created.
modifyDate	6 of int32	Month, day, year, hour, minute, second that the associated cell was last changed.
creator	char*	User name of the process that created the associated cell.
cellFlag	uint32	See “cellFlag” on page 3-106.
cellClass	int32	An 8-bit field that indicates the cell classes that are allowed in a row (zero means any class is allowed; nonzero means only cells with the same cellClass are allowed).
logicalEquivalentClass	uint32	Logical equivalent class of the associated cell (zero means there are no other logical equivalents; nonzero means that all cells with the same value are logically equivalent).
cellPowerValue	int32	Power consumed by associated cell (in database units).

Table 3-8 CellInfo Object Field Descriptions (Continued)

Field	Output	Description
subTileFlag	int32	Identifies the subtiles contained with the associated cell.
clientObj	int32	Client object.
creatorFlag	int32	Cell creator flag.
flag	int32	Flag for planet application.
cellClockFrequency	int32	Cell clock frequency.
cellPowerConsumption	int32	Cell power consumption.
cellVoltage	int32	Cell voltage.

The prototypes for the CellInfo accessor functions are

```
int MWXDb_Get_CellInfo_name (MWXCellId_t cellId, MWXObjId_t
    objId, char **string);

int MWXDb_Get_CellInfo_creationDate (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *hour, int32 *minute, int32
    *second, int32 *month, int32 *day, int32 *year);

int MWXDb_Get_CellInfo_modifyDate (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *hour, int32 *minute, int32
    *second, int32 *month, int32 *day, int32 *year);

int MWXDb_Get_CellInfo_creator (MWXCellId_t cellId,
    MWXObjId_t objId, char **creator);

int MWXDb_Get_CellInfo_cellFlag (MWXCellId_t cellId,
    MWXObjId_t objId, uint32 *cellFlag);

int MWXDb_Get_CellInfo_cellClass (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *cellClass);

int MWXDb_Get_CellInfo_logicalEquivalentClass (MWXCellId_t
    cellId, MWXObjId_t objId, uint32 *logE.Class);

int MWXDb_Get_CellInfo_subTileFlag (MWXCellId_t cellId,
    MWXObjId_t objId, int *subTileFlag);

int MWXDb_Get_CellInfo_clientObj (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *clientObj);

int MWXDb_Get_CellInfo_creatorFlag (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);
```

```

int MWXDb_Get_CellInfo_flag (MWXCellId_t cellId, MWXObjId_t
    objId, uint32 *value);

int MWXDb_Get_CellInfo_cellPowerValue (MWXCellId_t cellId,
    MWXObjId_t objId, int *cellPowerValue);

int MWXDb_Get_CellInfo_cellClockFrequency (MWXCellId_t
    cellId, MWXObjId_t objId, int32 *value);

int MWXDb_Get_CellInfo_cellPowerConsumption (MWXCellId_t
    cellId, MWXObjId_t objId, int32 *value);

int MWXDb_Get_CellInfo_cellVoltage (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

```

CellInst Object

The field descriptions for the CellInst object are listed in [Table 3-9](#).

Table 3-9 CellInst Object Field Descriptions

Field	Output	Description
name	char*	Hierarchical cell instance name.
position	MWXPoint_t*	Specifies x/y-coordinate pair defining location.
transformCode	int32	See “transformCode” on page 3-111 .
cellInstMasterId	MWXObjId_t	Database identifier of the CellInstMaster object associated with the CellInst object.
cellFlag	uint32	See “cellFlag” on page 3-106 .
flag1	int32	Status flag for placement and ECO with values: bit 0 - placed bit 1 - fixed bit 2 - soft fixed bit 3 - ECO delete bit 4 - ECO add bit 5 - ECO ignore bit 6 - GCTS add

Table 3-9 CellInst Object Field Descriptions (Continued)

Field	Output	Description
flag2	int32	Status flag for specific cell instance attributes with values: bit 0 - power cell instance up to date bit 1 - cell instance traced bit 7 - PR cell instance application mark
flag3	int32	Status flag for timing NoTouch with values: 0 - free 1 - don't touch

The prototypes for the CellInst accessor functions are

```
int MWXDb_Get_CellInst_name (MWXCellId_t cellId, MWXObjId_t
    objId, char **string);

int MWXDb_Get_CellInst_position (MWXCellId_t cellId,
    MWXObjId_t objId, MWXPoint_t **point);

int MWXDb_Get_CellInst_transformCode (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_CellInst_cellInstMasterId (MWXCellId_t
    cellId, MWXObjId_t objId, MWXObjId_t *nObjId);

int MWXDb_Get_CellInst_cellFlag (MWXCellId_t cellId,
    MWXObjId_t objId, uint32 *value);

int MWXDb_Get_CellInst_flag1 (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_CellInst_flag2 (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_CellInst_flag3 (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);
```

Additional functions related to CellInst are

- [MWXDb_Add_CellInstance_ToCellRegion](#)
- [MWXDb_Add_CellInst_ToPlanGroup_ByName](#)
- [MWXDb_Create_CellInstance](#)
- [MWXDb_Delete_CellInstance_FromCellRegion](#)
- [MWXDb_Place_CellInstanceInLegalLoc](#)

- [MWXDb_Get_CellInstanceBBox](#)
- [MWXDb_Get_CellInstance_ByName](#)
- [MWXDb_Get_CellInstCellType_Bit](#)
- [MWXDb_Get_CellInstFromCellInstMaster](#)
- [MWXDb_Get_CellInst_PowerDelayScale](#)
- [MWXDb_Get_CellInst_PowerId](#)
- [MWXDb_Get_CellInst_PowerSupplyVoltage](#)
- [MWXDb_Get_CellInst_PowerValue](#)
- [MWXDb_Get_CellInst_PowerTemperature](#)
- [MWXDb_Set_CellInst_PowerDelayScale](#)
- [MWXDb_Set_CellInst_PowerSupplyVoltage](#)
- [MWXDb_Set_CellInst_PowerTemperature](#)
- [MWXDb_Set_CellInst_PowerValue](#)
- [MWXDb_Set_CellInstanceECOStatus_ByName](#)
- [MWXDb_Set_CellInstanceFixStatus_ByName](#)
- [MWXDb_Set_CellLayoutSubType](#)
- [MWXDb_Set_StdCell](#)

CellInstArray Object

The field descriptions for the CellInstArray object are listed in [Table 3-10](#).

Table 3-10 CellInstArray Object Field Descriptions

Field	Output	Description
name	char*	Name of the array.
position	MWXPoint_t*	Specification of the x/y-coordinate pair defining location.
transformCode	int32	See “transformCode” on page 3-111.

Table 3-10 CellInstArray Object Field Descriptions (Continued)

Field	Output	Description
XduplicateN	uint32	Number of cell instances in the horizontal direction.
YduplicateN	uint32	Number of cell instances in the vertical direction.
xDisplacement	int32	Horizontal distance between cell instances.
yDisplacement	int32	Vertical distance between cell instances.
cellInstMasterId	MWXObjId_f	Database identifier of CellInstMaster object for cell instances contained in the array.
bbox	4 of int32	Two x/y coordinate pairs representing the lower-left and upper-right corners of the bounding box of the array.
cellName	char*	Name of the cell instantiated in the array.
viewName	char*	View of the cell instantiated in the array.
cellVersion	int32	Version number of the cell instantiated in the array.
cellFlag	uint32	See “cellFlag” on page 3-106 .
flag1	int32	Status flag for placement and ECO with values: bit 0 - placed bit 1 - fixed bit 2 - soft fixed bit 3 - ECO delete bit 4 - ECO add bit 5 - ECO ignore bit 6 - GCTS add
flag2	int32	Status flag for specific cell instance attributes with values: bit 0 - power cell instance up to date bit 1 - cell instance traced bit 7 - PR cell instance application mark

Table 3-10 *CellInstArray Object Field Descriptions (Continued)*

Field	Output	Description
flag3	int32	Status flag for timing NoTouch with values: 0 - free 1 - don't touch

The prototypes for the CellInstArray accessor functions are

```
int MWXDb_Get_CellInstArray_name (MWXCellId_t cellId,
    MWXObjId_t objId, char **string);

int MWXDb_Get_CellInstArray_position (MWXCellId_t cellId, MWXObjId_t objId,
    MWXPoint_t **point);

int MWXDb_Get_CellInstArray_transformCode (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_CellInstArray_XduplicateN (MWXCellId_t cellId,
    MWXObjId_t objId, uint32 *value);

int MWXDb_Get_CellInstArray_YduplicateN (MWXCellId_t cellId,
    MWXObjId_t objId, uint32 *value);

int MWXDb_Get_CellInstArray_xDisplacement (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_CellInstArray_yDisplacement (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_CellInstArray_cellInstMasterId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *nObjId);

int MWXDb_Get_CellInstArray_bbox (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *left, int32 *bottom, int32 *right, int32 *top);

int MWXDb_Get_CellInstArray_cellName (MWXCellId_t cellId,
    MWXObjId_t objId, char **string);

int MWXDb_Get_CellInstArray_viewName (MWXCellId_t cellId,
    MWXObjId_t objId, char **string);

int MWXDb_Get_CellInstArray_cellVersion (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_CellInstArray_cellFlag (MWXCellId_t cellId,
    MWXObjId_t objId, uint32 *value);

int MWXDb_Get_CellInstArray_flag1 (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_CellInstArray_flag2 (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);
```

```
int MWXDb_Get_CellInstArray_flag3 (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);
```

CellInstMaster Object

The field descriptions for the CellInstMaster object are listed in [Table 3-11](#).

Table 3-11 CellInstMaster Object Field Descriptions

Field	Output	Description
name	char*	Name of the associated cell in the form, <i>Cellname.viewname</i> .
bbox	4 of int32	Two x/y-coordinate pairs representing the lower-left and upper-right corners of the bounding box of the associated cell.
cellName	char*	Cell name of associated cell.
viewName	char*	View name of associated cell.
cellClass	int32	An 8-bit field that indicates the cell classes that are allowed in a row (zero means any class is allowed; nonzero means only cells with the same cellClass are allowed).
logicalEquivalentClass	uint32	Logical equivalent class of the associated cell (zero means there are no other logical equivalents; nonzero means that all cells with the same value are logically equivalent).
numberOfPin	int32	Number of pins owned by the associated cell.
cellVersion	int32	Version of the associated cell.
cellFlag	uint32	See “cellFlag” on page 3-106 .
numberOfCellInst	uint32	Number of instantiations of the associated cell.
numberOfPort	uint32	Number of ports owned by the associated cell.
modifyDate	uint32	Month, day, year, hour, minute, and second that the associated cell was modified.

The prototypes for the CellInstMaster accessor functions are

```

int MWXDb_Get_CellInstMaster_name (MWXCellId_t cellId,
    MWXObjId_t objId, char **string);

int MWXDb_Get_CellInstMaster_bbox (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *left, int32 *bottom, int32 *right, int32 *top);

int MWXDb_Get_CellInstMaster_cellName (MWXCellId_t cellId,
    MWXObjId_t objId, char **string);

int MWXDb_Get_CellInstMaster_viewName (MWXCellId_t cellId,
    MWXObjId_t objId, char **string);

int MWXDb_Get_CellInstMaster_cellClass (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_CellInstMaster_logicalEquivalentClass (MWXCellId_t cellId,
    MWXObjId_t objId, uint32 *value);

int MWXDb_Get_CellInstMaster_numberOfPin (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_CellInstMaster_cellVersion (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_CellInstMaster_cellFlag (MWXCellId_t cellId,
    MWXObjId_t objId, uint32 *value);

int MWXDb_Get_CellInstMaster_numberOfCellInst (MWXCellId_t cellId,
    MWXObjId_t objId, uint32 *value);

int MWXDb_Get_CellInstMaster_numberOfPort (MWXCellId_t cellId,
    MWXObjId_t objId, uint32 *value);

int MWXDb_Get_CellInstMaster_modifyDate (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *hour, int32 *minute, int32 *second,
    int32 *month, int32 *day, int32 *year);

```

Additional functions related to CellInstMaster are

- [MWXDb_Display_DiodePinInstMaster_FromDiodePortInstMaster](#)
- [MWXDb_Get_CellInstFromCellInstMaster](#)
- [MWXDb_Purge_CellInstMaster](#)

CellRegion Object

The field descriptions for the CellRegion object are listed in [Table 3-12](#).

Table 3-12 CellRegion Object Field Descriptions

Field	Output	Description
name	char*	Cell region name
bbox	4 of int32	Two x/y-coordinate pairs representing the lower-left and upper-right corners of the bounding box of the rectangle
utilization	int32	Target hundredth of percentage
cellInstGroupId	int32	Cell instance group ID
cellRowPattern	int32	Cell row pattern for double back and flip

The prototypes for the CellRegion accessor functions are

```
int MWXDb_Get_CellRegion_name (MWXCellId_t cellId,  
                               MWXObjId_t objId, char **string);  
  
int MWXDb_Get_CellRegion_bbox ((MWXCellId_t cellId,  
                                MWXObjId_t objId, int32 *left, int32 *bottom, int32 *right, int32 *top));  
  
int MWXDb_Get_CellRegion_utilization ((MWXCellId_t cellId,  
                                       MWXObjId_t objId, int32 *value));  
  
int MWXDb_Get_CellRegion_cellInstGroupId (MWXCellId_t cellId,  
                                           MWXObjId_t objId, MWXObjId_t *cellInstGroupId);  
  
int MWXDb_Get_CellRegion_cellRowPattern (MWXCellId_t cellId,  
                                          MWXObjId_t objId, int32 *cellRowPattern);
```

CellRow Object

The field descriptions for the CellRow object are listed in [Table 3-13](#).

Table 3-13 CellRow Object Field Descriptions

Field	Output	Description
bbox	4 of int32	Two x/y-coordinate pairs representing the lower-left and upper-right corners of the bounding box of the rectangle
allowablePattern	int32 *	The allowable patterns
cellRatio	int32	Cell area to row area ratio
index	int32	Sequential index
linkCellRowId	int32	Link cell row ID

The prototypes for the CellRow accessor functions are

```
int MWXDb_Get_CellRow_bbox ((MWXCellId_t cellId, MWXObjId_t
    objId, int32 *left, int32 *bottom, int32 *right, int32 *top));

int MWXDb_Get_CellRow_allowablePattern ((MWXCellId_t cellId, MWXObjId_t
    objId, int32 *numOfElm, int32 **value));

int MWXDb_Get_CellRow_cellRatio (MWXCellId_t cellId, MWXObjId_t
    objId, int *cellRatio);

int MWXDb_Get_CellRow_index (MWXCellId_t cellId, MWXObjId_t
    objId, int *index);

int MWXDb_Get_CellRow_linkCellRowId (MWXCellId_t cellId, MWXObjId_t
    objId, MWXObjId_t *linkCellRowId);
```

CLFRecord Object

The field descriptions for the CLFRecord object are listed in [Table 3-14](#).

Table 3-14 CLFRecord Object Field Descriptions

Field	Output	Description
operation	int32	Operation

Table 3-14 CLFRecord Object Field Descriptions

Field	Output	Description
dataLength	int32	Data length
data	int32	Array of data

The prototypes for the CLFRecord accessor functions are

```
int MWXDb_Get_CLFRecord_operation (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_CLFRecord_dataLength (MWXCellId_t cellId,
    MWXObjId_t objId, uint32 *value);

int MWXDb_Get_CLFRecord_data (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *numOfElm, int32 **value);
```

ClkOptData Object

The field descriptions for the ClkOptData object are listed in [Table 3-15](#).

Table 3-15 ClkOptData Object Field Descriptions

Field	Output	Description
riseMin	int32	Rise minimum value
riseMax	int32	Rise maximum value
fallMin	int32	Fall minimum value
fallMax	int32*	Fall minimum value
level	int32	Level
phaseFlag	int32	Phase flag with values: 1 - noninvert rise 2 - noninvert fall 4 - invert rise 8 - invert fall
signature	int32	Signature

Table 3-15 *ClkOptData Object Field Descriptions (Continued)*

Field	Output	Description
statusFlag	int32	Flag indicating set sync pin or ignore pin with values: 1 - sync type 2 - ignore type 4 - CTS type

The prototypes for the ClkOptData accessor functions are

```
int MWXDb_Get_ClkOptData_invertDel (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *riseMin, int32 *riseMax, int32 *fallMin,
    int32 *fallMax);

int MWXDb_Get_ClkOptData_nonInvDel (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *riseMin, int32 *riseMax, int32 *fallMin,
    int32 *fallMax);

int MWXDb_Get_ClkOptData_level (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *level);

int MWXDb_Get_ClkOptData_phaseFlag (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *phaseFlag);

int MWXDb_Get_ClkOptData_signature (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *signature);

int MWXDb_Get_ClkOptData_statusFlag (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *statusFlag);
```

Contact Object

The field descriptions for the Contact object are listed in [Table 3-16](#).

Table 3-16 *Contact Object Field Descriptions*

Field	Output	Description
netId	MWXObjId_t	Database identifier of the net associated with the contact.
contactCodIdx	int32	Index into the contact table of the technology file (used to define contact characteristics).
transformCode	int32	See “transformCode” on page 3-111 .

Table 3-16 Contact Object Field Descriptions

Field	Output	Description
centerPosition	MWXPoint_t*	Specification of the x/y-coordinates at which the center of the contact is placed.
routeType	int32	See “routeType” on page 3-108 .

The prototypes for the Contact accessor functions are

```
int MWXDb_Get_Contact_netId (MWXCellId_t cellId, MWXObjId_t objId,
                             MWXObjId_t *nObjId);

int MWXDb_Get_Contact_contactCodeIdx (MWXCellId_t cellId,
                                       MWXObjId_t objId, int32 *value);

int MWXDb_Get_Contact_transformCode (MWXCellId_t cellId,
                                      MWXObjId_t objId, int32 *value);

int MWXDb_Get_Contact_centerPosition (MWXCellId_t cellId,
                                       MWXObjId_t objId, MWXPoint_t **point);

int MWXDb_Get_Contact_routeType (MWXCellId_t CellId, M
                                WXObjId_t objId, uchar routeType);
```

Additional functions related to Contact are

- [MWXDb_Create_Contact_WithRoutType](#)
- [MWXDb_Get_ContactGeom](#)
- [MWXDb_Get_DefaultContactByMetalLayers](#)
- [MWXDb_Get_DefaultContactByViaLayer](#)
- [MWXDb_Get_FatContactByMaskName](#)
- [MWXDb_Get_TechContactLayer](#)
- [MWXDb_Get_TechContactSize](#)

ContactArray Object

The field descriptions for the ContactArray object are listed in [Table 3-17](#).

Table 3-17 ContactArray Object Field Descriptions

Field	Output	Description
bbox	4 of int32	Two x/y-coordinate pairs representing the lower-left and upper-right corners of the bounding box of the array.
netId	MWXObjId_t	Database identifier of the net associated with the array.
contactCodIdx	int32	Index into the contact table of the technology file (used to define contact characteristics).
transformCode	int32	See “transformCode” on page 3-111 .
flag	int32	Zero if no bitmap is present, one otherwise.
position	MWXPoint_t*	Specification of the x/y-coordinates at which the center of the array is placed.
xDuplicate	int32	Number of contacts in the horizontal direction.
yDuplicate	int32	Number of contacts in the vertical direction.
xPitch	int32	Horizontal center-to-center spacing of the contacts.
yPitch	int32	Vertical center-to-center spacing of the contacts
routeType	int32	See “routeType” on page 3-108 .
bitmap	int32*	Represents the contact array in bitmap.

The prototypes for the ContactArray accessor functions are

```
int MWXDb_Get_ContactArray_bbox (MWXCellId_t cellId,  
    MWXObjId_t objId, int32 *left, int32 *bottom, int32 *right, int32 *top);  
  
int MWXDb_Get_ContactArray_netId (MWXCellId_t cellId,  
    MWXObjId_t objId, MWXObjId_t *nObjId);
```

```

int MWXDb_Get_ContactArray_contactCodeIdx (MWXCellId_t cellId,
      MWXObjId_t objId, int32 *value);

int MWXDb_Get_ContactArray_transformCode (MWXCellId_t cellId,
      MWXObjId_t objId, int32 *value);

int MWXDb_Get_ContactArray_flag (MWXCellId_t cellId,
      MWXObjId_t objId, int32 *value);

int MWXDb_Get_ContactArray_position (MWXCellId_t cellId,
      MWXObjId_t objId, MWXPoint_t **point);

int MWXDb_Get_ContactArray_xDuplicate (MWXCellId_t cellId,
      MWXObjId_t objId, int32 *value);

int MWXDb_Get_ContactArray_yDuplicate (MWXCellId_t cellId,
      MWXObjId_t objId, int32 *value);

int MWXDb_Get_ContactArray_xPitch (MWXCellId_t cellId,
      MWXObjId_t objId, int32 *value);

int MWXDb_Get_ContactArray_yPitch (MWXCellId_t cellId,
      MWXObjId_t objId, int32 *value);

int MWXDb_Get_ContactArray_routeType(MWXCellId_t CellId,
      MWXObjId_t objId, int32 routeType);

int MWXDb_Get_ContactArray_bitmap(MWXCellId_t CellId,
      MWXObjId_t objId, int32 **bitmap, int32 *numOfElem);

```

See also

- [MWXDb_Create_ContactArray_WithRoutType](#)

DelayTable Object

The field descriptions for the DelayTable object are listed in [Table 3-18](#).

Table 3-18 DelayTable Object Field Descriptions

Field	Output	Description
tableType	int32	See “ tableType ” on page 3-110.
index1Type	int32	Index type for one-dimensional table.
index2Type	int32	Index type for two-dimensional table.
index3Type	int32	Index type for three-dimensional table.
lengthIndex1	int32	Length of index 1.

Table 3-18 *DelayTable Object Field Descriptions (Continued)*

Field	Output	Description
lengthIndex2	int32	Length of index 2.
lengthIndex3	int32	Length of index 3.
valueOrValueTable	int32	Scalar type table or one-dimensional table.

The prototypes for the DelayTable accessor functions are

```
int MWXDb_Get_DelayTable_tableType (MWXCellId_t cellId,
                                     MWXObjId_t objId, int32 *tableType);

int MWXDb_Get_DelayTable_index1Type (MWXCellId_t cellId,
                                     MWXObjId_t objId, int32 *index1Type);

int MWXDb_Get_DelayTable_index2Type (MWXCellId_t cellId,
                                     MWXObjId_t objId, int32 *index2Type);

int MWXDb_Get_DelayTable_index3Type (MWXCellId_t cellId,
                                     MWXObjId_t objId, int32 *index3Type);

int MWXDb_Get_DelayTable_lengthIndex1 (MWXCellId_t cellId,
                                       MWXObjId_t objId, int32 *lengthIndex1);

int MWXDb_Get_DelayTable_lengthIndex2 (MWXCellId_t cellId,
                                       MWXObjId_t objId, int32 *lengthIndex2);

int MWXDb_Get_DelayTable_lengthIndex3 (MWXCellId_t cellId,
                                       MWXObjId_t objId, int32 *lengthIndex3);

int MWXDb_Get_DelayTable_valueOrValueTable (MWXCellId_t cellId,
                                             MWXObjId_t objId, int32 *valueOrValueTable);
```

ErrorComponent Object

An error component associates a net and a rectangular area with an ErrorObj. An ErrorObj might have several ErrorComponents, whereas an ErrorComponent can have only one ErrorObj.

This object is not persistent, so when a cell is closed, it is not saved as part of the cell and is lost.

The field descriptions for the ErrorComponent object are listed in [Table 3-19](#).

Table 3-19 ErrorComponent Object Field Descriptions

Field	Output	Description
netId	MWXObjId_t	Database identifier of the net associated with the error component
nodeNumber	int	Integer value displayed by verification tools for the error component
errorTypeId	MWXObjId_t	Database identifier of the ErrorType object to which the ErrorComponent object belongs
bbox	4 of int32	Two x/y-coordinate pairs representing the lower-left and upper-right corners of the bounding box of the rectangle

The prototypes for the ErrorComponent accessor functions are

```
int MWXDb_Get_ErrorComponent_netId (MWXCellId_t cellId,  
    MWXObjId_t objId, MWXObjId_t *nObjId);  
  
int MWXDb_Get_ErrorComponent_nodeNumber (MWXCellId_t cellId,  
    MWXObjId_t objId, int32 *value);  
  
int MWXDb_Get_ErrorComponent_errorObjId (MWXCellId_t cellId,  
    MWXObjId_t objId, MWXObjId_t *nObjId);  
  
int MWXDb_Get_ErrorComponent_bbox (MWXCellId_t cellId,  
    MWXObjId_t objId, int32 *left, int32 *bottom, int32 *right, int32 *top);
```

See also

- [MWXDb_Create_ErrorComponents](#)

ErrorObj Object

An error object is an occurrence of a particular type of error described by an ErrorType object. This object is not persistent, so when a cell is closed, it is not saved as part of the cell and is lost.

The field descriptions for the ErrorObj object are listed in [Table 3-20](#).

Table 3-20 ErrorObj Object Field Descriptions

Field	Output	Description
info	char*	Text message associated with the error type and reported by the verification tool
status	uint16	Currently unused
bbox	4 of int32	Two x/y-coordinate pairs representing the lower-left and upper-right corners of the bounding box of the rectangle
numberOfComponent	uint32	A count of the error component records associated with the ErrorObj object
errorTypeId	MWXObjId_t	Database identifier of the ErrorType object to which the ErrorObj object belongs

The prototypes for the ErrorObj accessor functions are

```
int MWXDb_Get_ErrorObj_info (MWXCellId_t cellId,
                             MWXObjId_t objId, char **string);

int MWXDb_Get_ErrorObj_status (MWXCellId_t cellId,
                              MWXObjId_t objId, int32 *value);

int MWXDb_Get_ErrorObj_bbox (MWXCellId_t cellId,
                             MWXObjId_t objId, int32 *left, int32 *bottom, int32 *right, int32 *top);

int MWXDb_Get_ErrorObj_numberOfComponent (MWXCellId_t cellId,
                                           MWXObjId_t objId, int32 *value);

int MWXDb_Get_ErrorObj_errorTypeId (MWXCellId_t cellId,
                                     MWXObjId_t objId, MWXObjId_t *nObjId);
```

Additional functions related to ErrorObj are

- [MWXDb_Create_ErrorComponents](#)
- [MWXDb_Create_ErrorObj](#)
- [MWXDb_Get_ErrorComponent_errorObjId](#) (see [ErrorComponent Object](#))
- [MWXDb_Get_ErrorType_numberOfErrorObj](#) (see [ErrorType Object](#))

- [MWXDb_Set_ErrorObjStatus](#)

ErrorType Object

An error type is a template or a kind of error, much like a cell is to an instance. This object is not persistent, so when a cell is closed it is not saved as part of the cell and is lost.

The field descriptions for the ErrorType object are listed in [Table 3-21](#).

Table 3-21 ErrorType Object Field Descriptions

Field	Output	Description
info	char*	Text message associated with the error type and reported by the verification tool
level	uchar	Severity level of the error: 0 = informational 1 = warning 2 = error 3 = severe error
status	uint16	Currently unused
numberOfErrorObj	uint32	A count of the error object records associated with the ErrorType object
bbox	4 of int32	Two x/y-coordinate pairs representing the lower-left and upper-right corners of the bounding box of the rectangle
summary	char*	Text summary that appears on display; maximum of 15 characters

The prototypes for the ErrorType accessor functions are

```
int MWXDb_Get_ErrorType_info (MWXCellId_t cellId,
                             MWXObjId_t objId, char **string);

int MWXDb_Get_ErrorType_level (MWXCellId_t cellId,
                              MWXObjId_t objId, uint32 *value);

int MWXDb_Get_ErrorType_status (MWXCellId_t cellId,
                               MWXObjId_t objId, uint32 *value);

int MWXDb_Get_ErrorType_numberOfErrorObj (MWXCellId_t cellId,
                                           MWXObjId_t objId, int32 *value);
```

```
int MWXDb_Get_ErrorType_bbox (MWXCellId_t cellId,
                             MWXObjId_t objId, int32 *left, int32 *bottom, int32 *right, int32 *top);
```

```
int MWXDb_Get_ErrorType_summary (MWXCellId_t cellId,
                                 MWXObjId_t objId, char **string);
```

Additional functions related to `ErrorType` are

- [MWXDb_Create_ErrorObj](#)
- [MWXDb_Create_ErrorType](#)
- [MWXDb_Get_ErrorObj_errorTypeId](#) (see [ErrorObj Object](#))
- [MWXDb_Set_ErrorTypeStatus](#)

ExpCellInst Object

The field descriptions for the `ExpCellInst` object are listed in [Table 3-22](#).

Table 3-22 ExpCellInst Object Field Descriptions

Field	Output	Description
flag1	int32	Status flag for placement and ECO with values: bit 0 - placed bit 1 - fixed bit 2 - soft fixed bit 3 - ECO delete bit 4 - ECO add bit 5 - ECO ignore bit 6 - GCTS add
flag2	int32	Status flag for specific cell instance attributes with values: bit 0 - power cell instance up to date bit 1 - cell instance traced bit 7 - PR cell instance application mark
flag3	int32	Status flag for timing NoTouch with values: 0 - free 1 - don't touch

The prototypes for the `ExpCellInst` accessor functions are

```

int MWXDb_Get_ExpCellInst_flag1 (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_ExpCellInst_flag2 (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_ExpCellInst_flag3 (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

```

ExpPinInst Object

The field descriptions for the ExpPinInst object are listed in [Table 3-23](#).

Table 3-23 ExpPinInst Object Field Descriptions

Field	Output	Description
mustJoinClass	int32	Array of must join class
statusFlag	int32	Array of status flag with values: bit 0 - side constrained bit 1 - order constrained bit 2 - location constrained bit 3 - pin manual created bit 4 - right-side pin bit 5 - left-side pin bit 6 - Top pin bit 7 - Bottom pin

The prototypes for the ExpPinInst accessor functions are

```

int MWXDb_Get_ExpPinInst_mustJoinClass (MWXCellId_t cellId,
    MWXObjId_t objId, int *number, uint32 **value);

int MWXDb_Get_ExpPinInst_statusFlag1 (MWXCellId_t cellId,
    MWXObjId_t objId, int *number, int32 **value);

```

ExpPortInst Object

The field description for the ExpPortInst object is listed in [Table 3-24](#).

Table 3-24 ExpPortInst Object Field Description

Field	Output	Description
constraints	int32	Array of constraints

The prototype for the ExpPortInst accessor function is

```
int MWXDb_Get_ExpPortInst_constraints (MWXCellId_t cellId,  
                                       MWXObjId_t objId, int32 *numOfElm, int32 **value);
```

ExtraPin Object

The object and field descriptions for the ExtraPin database record are listed in [Table 3-25](#).

Table 3-25 ExtraPin Object Field Descriptions

Field	Output	Description
name	char*	Name of related ExtraPort
extraPortId	MWXObjId_t	Database identifier of related ExtraPort
bbox	4 of int32	Two x/y coordinate pairs representing the lower-left and upper-right corners of the bounding box of the ExtraPin
layer	int32	Number of the layer on which the ExtraPin resides.
dataType	int32	Number of the dataType which the ExtraPin is assigned.
numberOfPoint	uint32	Number of points used to define the ExtraPin.
points	uint32, MWXPoint_t array	ExtraPin and an array of the x/y coordinate pairs.

The function prototypes that correspond to the field names (listed in [Table 3-25](#)) for the ExtraPin database record are as follows:

```
int MWXDb_Get_ExtraPin_name (MWXCellId_t cellId, MWXObjId_t objId,
    char **string);

int MWXDb_Get_ExtraPin_extraPortId (MWXCellId_t cellId, MWXObjId_t objId,
    MWXObjId_t *nObjId);

int MWXDb_Get_ExtraPin_bbox (MWXCellId_t cellId, MWXObjId_t objId,
    int32 *left, int32 *bottom, int32 *right, int32 *top);

int MWXDb_Get_ExtraPin_layer (MWXCellId_t cellId, MWXObjId_t objId,
    int32 *value);

int MWXDb_Get_ExtraPin_dataType (MWXCellId_t cellId, MWXObjId_t objId,
    int32 *value);

int MWXDb_Get_ExtraPin_numberOfPoint (MWXCellId_t cellId, MWXObjId_t objId,
    uint32 *value);

int MWXDb_Get_ExtraPin_points (MWXCellId_t cellId, MWXObjId_t objId,
    int32 *numOfPoint, MWXPoint_t **points);
```

ExtraPinInstMaster Object

The object and field descriptions for the ExtraPinInstMaster database record are listed in [Table 3-26](#).

Table 3-26 ExtraPortInstMaster Object Field Descriptions

Field	Output	Description
name	char*	Name of related ExtraPortInstMaster
cellInstMasterId	MWXObjId_t	Database identifier of related cell instance master.
bbox	4 of int32	Two x/y coordinate pairs representing the lower-left and upper-right corners of the bounding box of the ExtraPin
layer	int32	Number of the layer on which the ExtraPin resides.
dataType	int32	Number of the dataType which the ExtraPin is assigned.

Table 3-26 *ExtraPortInstMaster Object Field Descriptions*

Field	Output	Description
numberOfPoint	uint32	Number of points used to define the ExtraPin.
points	uint32, MWXPoint_t array	ExtraPin and an array of the x/y coordinate pairs.

The function prototypes that correspond to the field names (listed in [Table 3-26](#)) for the ExtraPinInstMaster database record are as follows:

```
int MWXDb_Get_ExtraPinInstMaster_name (MWXCellId_t cellId, MWXObjId_t objId,
    char **string);

int MWXDb_Get_ExtraPinInstMaster_cellInstMasterId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *nObjId);

int MWXDb_Get_ExtraPinInstMaster_extraPortInstMasterId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *nObjId);

int MWXDb_Get_ExtraPinInstMaster_bbox (MWXCellId_t cellId, MWXObjId_t objId,
    int32 *left, int32 *bottom, int32 *right, int32 *top);

int MWXDb_Get_ExtraPinInstMaster_layer (MWXCellId_t cellId, MWXObjId_t objId,
    int32 *value);

int MWXDb_Get_ExtraPinInstMaster_dataType (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_ExtraPinInstMaster_numberOfPoint (MWXCellId_t cellId,
    MWXObjId_t objId, uint32 *value);

int MWXDb_Get_ExtraPinInstMaster_points (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *numOfPoint, MWXPoint_t **points);
```

ExtraPort Object

The object and field descriptions for the ExtraPort database record are listed in [Table 3-27](#).

Table 3-27 *ExtraPort Object Field Descriptions*

Field	Output	Description
name	char*	Name of related ExtraPort
type	uint32	Type of port like P/G and tie-off

Table 3-27 ExtraPort Object Field Descriptions

Field	Output	Description
numberOfExtraPin	uint32	Number of related ExtraPin objects

The function prototypes that correspond to the field names (listed in [Table 3-27](#)) for the ExtraPort database record are as follows:

```
int MWXDb_Get_ExtraPort_name (MWXCellId_t cellId, MWXObjId_t objId,
    char **string);

int MWXDb_Get_ExtraPort_type (MWXCellId_t cellId, MWXObjId_t objId,
    uint32 *value);

int MWXDb_Get_ExtraPort_numberOfExtraPin (MWXCellId_t cellId,
    MWXObjId_t objId, uint32 *value);
```

ExtraPortInst Object

The object and field descriptions for the ExtraPortInst database record are listed in [Table 3-28](#).

Table 3-28 ExtraPortInst Object Field Descriptions

Field	Output	Description
netId	MWXObjId_t	Database identifier of related net.
cellInstId	MWXObjId_t	Database identifier of related cellInst.
extraPortInstMasterId	MWXObjId_t	Database identifier of related ExtraPortInstMaster

The function prototypes that correspond to the field names (listed in [Table 3-28](#)) for the ExtraPortInst database record are as follows:

```
int MWXDb_Get_ExtraPortInst_netId (MWXCellId_t cellId, MWXObjId_t objId,
    MWXObjId_t *nObjId);

int MWXDb_Get_ExtraPortInst_cellInstId (MWXCellId_t cellId, MWXObjId_t objId,
    MWXObjId_t *nObjId);

int MWXDb_Get_ExtraPortInst_extraPortInstMasterId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *nObjId);
```

ExtraPortInstMaster Object

The object and field descriptions for the ExtraPortInstMaster database record are listed in [Table 3-29](#).

Table 3-29 ExtraPortInstMaster Object Field Descriptions

Field	Output	Description
name	char*	Name of related ExtraPortInstMaster
cellInstMasterId	MWXObjId_t	Database identifier of related cell instance master.
extraPortInstMasterId	MWXObjId_t	Database identifier of related ExtraPortInstMaster
numberOfExtraPinInstMaster	uint32	Number of ExtraPinInstMaster
extraPortId	MWXObjId_t	Database identifier of related ExtraPort

The function prototypes that correspond to the field names (listed in [Table 3-29](#)) for the ExtraPortInstMaster database record are as follows:

```
int MWXDb_Get_ExtraPortInstMaster_name (MWXCellId_t cellId, MWXObjId_t objId,
char **string);

int MWXDb_Get_ExtraPortInstMaster_cellInstMasterId (MWXCellId_t cellId,
MWXObjId_t objId, MWXObjId_t *nObjId);

int MWXDb_Get_ExtraPortInstMaster_type (MWXCellId_t cellId, MWXObjId_t objId,
uint32 *value);

int MWXDb_Get_ExtraPortInstMaster_numberOfExtraPinInstMaster (
MWXCellId_t cellId, MWXObjId_t objId, uint32 *value);

int MWXDb_Get_ExtraPortInstMaster_extraPortId (MWXCellId_t cellId,
MWXObjId_t objId, MWXObjId_t *nObjId);
```

Glink Object

A point-to-point connection, usually vertical or horizontal, but can be diagonal.

The field descriptions for the Glink object are listed in [Table 3-30](#).

Table 3-30 Glink Object Field Descriptions

Field	Output	Description
netId	MWXObjectId_t	Database identifier of net within the owning cell to which Glink is attached.
layer	int32	Number of the layer on which the Glink resides. (Layer numbers appear in the technology file.)
dataType	int32	Data type of the Glink object.
routeType	int32	See “routeType” on page 3-108.
wireMasterIdx	int32	Index value of associated WireMaster object. (See index field in Table 3-89 on page 3-102).
point1	MWXPoint_t*	Specification of the x/y-coordinates of the first point of the Glink object.
point2	MWXPoint_t*	Specification of the x/y-coordinates of the second point of the Glink object.

The prototypes for the Glink accessor functions are

```
int MWXDb_Get_Glink_netId (MWXCellId_t cellId,  
    MWXObjId_t objId, MWXObjId_t * netId);  
  
int MWXDb_Get_Glink_layer (MWXCellId_t cellId, M  
    WXObjId_t objId, uint32 *layer);  
  
int MWXDb_Get_Glink_routetype (MWXCellId_t cellId,  
    MWXObjId_t objId, uint32 *routetype);  
  
int MWXDb_Get_Glink_wireMasterIdx (MWXCellId_t cellId,  
    MWXObjId_t objId, int32 *wireMasterIdx.);  
  
int MWXDb_Get_Glink_point1 (MWXCellId_t cellId,  
    MWXObjId_t objId, MWXPoint_t ** point1);  
  
int MWXDb_Get_Glink_point2 (MWXCellId_t cellId,  
    MWXObjId_t objId, MWXPoint_t ** point2);
```

See also

- [MWXDb_Create_Glink](#)

Group Object

The field descriptions for the Group object are listed in [Table 3-31](#)

Table 3-31 Group Object Field Descriptions

Field	Output	Description
groupMembers	MWXObjId_t*	Array of all objects in this group
groupSignature	int32	Group signature
name	char*	Group name

Note:

Groups are a special case without the usual `MWXDb_Get_<Object>_<field>` accessor functions.

Functions related to Group are

- [MWXDb_Add_Object_ToGroup](#)
- [MWXDb_Create_Group](#)
- [MWXDb_Get_GroupMember](#)
- [MWXDb_Get_Group_ByName](#)
- [MWXDb_Is_Object_InGroup](#)
- [MWXDb_Purge_Group_ByName](#)
- [MWXDb_Remove_Object_FromGroup](#)

HierCellInst Object

The object and field descriptions for the HierCellInst database record are listed in [Table 3-32](#).

Table 3-32 HierCellInst Object Field Descriptions

Field	Output	Description
name	char*	Name of hierarchical cell instance.

Table 3-32 HierCellInst Object Field Descriptions

Field	Output	Description
hierCellInstMasterId	MWXObjId_t	Database identifier of related hierarchical cell instance master.

The function prototypes that correspond to the field names (listed in [Table 3-32](#)) for the HierCellInst database record are as follows:

```
int MWXDb_Get_HierCellInst_name (MWXCellId_t cellId, MWXObjId_t objId,
    char **string);

int MWXDb_Get_HierCellInst_hierCellInstMasterId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *nObjId);

int MWXDb_Set_HierCellInst_name (MWXCellId_t cellId, MWXObjId_t objId,
    char *buffer);

int MWXDb_Set_HierCellInst_hierCellInstMasterId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t nObjId);
```

HierCellInstMaster Object

The object and field descriptions for the HierCellInstMaster database record are listed in [Table 3-33](#).

Table 3-33 HierCellInstMaster Object Field Descriptions

Field	Output	Description
name	char*	Name of hierarchical cell instance master.

The function prototypes that correspond to the field names (listed in [Table 3-33](#)) for the HierCellInstMaster database record are as follows:

```
int MWXDb_Get_HierCellInstMaster_name (MWXCellId_t cellId, MWXObjId_t objId,
    char **string);

int MWXDb_Set_HierCellInstMaster_name (MWXCellId_t cellId, MWXObjId_t objId,
    char *buffer);
```

HierPortInst Object

The object and field descriptions for the HierPortInst database record are listed in [Table 3-34](#).

Table 3-34 HierPortInst Object Field Descriptions

Field	Output	Description
name	char*	Name of hierarchical cell instance master.

The function prototypes that correspond to the field names (listed in [Table 3-34](#)) for the HierPortInst database record are as follows:

```
int MWXDb_Get_HierPortInst_hierPortInstMasterId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *nObjId);

int MWXDb_Get_HierPortInst_hierCellInstId (MWXCellId_t cellId, MWXObjId_t
    objId, MWXObjId_t *nObjId);

int MWXDb_Get_HierPortInst_hierNet (MWXCellId_t cellId, MWXObjId_t objId,
    MWXObjId_t *netId1, MWXObjId_t *netId2);

int MWXDb_Get_HierPortInst_flag (MWXCellId_t cellId, MWXObjId_t objId,
    uint32 *value);

int MWXDb_Set_HierPortInst_hierPortInstMasterId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t nObjId);

int MWXDb_Set_HierPortInst_hierCellInstId (MWXCellId_t cellId, MWXObjId_t
    objId, MWXObjId_t nObjId);

int MWXDb_Set_HierPortInst_hierNet (MWXCellId_t cellId, MWXObjId_t objId,
    MWXObjId_t netId1, MWXObjId_t netId2);

int MWXDb_Set_HierPortInst_flag (MWXCellId_t cellId, MWXObjId_t objId,
    uint32 value);
```

HierNet Object

The object and field descriptions for the HierNet database record are listed in [Table 3-35](#).

Table 3-35 HierNet Object Field Descriptions

Field	Output	Description
name	char*	Name of hierarchical net.
numberOfHierPortInst	uint32	Number of hierarchical port instance.
flag	uint32	General purpose bitmask flag.

The function prototypes that correspond to the field names (listed in [Table 3-35](#)) for the HierNet database record are as follows:

```
int MWXDb_Get_HierNet_name (MWXCellId_t cellId, MWXObjId_t objId,
    char **string);

int MWXDb_Get_HierNet_numberOfHierPortInst (MWXCellId_t cellId, MWXObjId_t
    objId, uint32 *value);

int MWXDb_Get_HierNet_flag (MWXCellId_t cellId, MWXObjId_t objId,
    uint32 *value);

int MWXDb_Set_HierNet_name (MWXCellId_t cellId, MWXObjId_t objId,
    char *buffer);

int MWXDb_Set_HierNet_numberOfHierPortInst (MWXCellId_t cellId, MWXObjId_t
    objId, uint32 value);

int MWXDb_Set_HierNet_flag (MWXCellId_t cellId, MWXObjId_t objId,
    uint32 value);
```

HierPortVectorInfo Object

The object and field descriptions for the HierPortVectorInfo database record are listed in [Table 3-36](#).

Table 3-36 HierPortVectorInfo Object Field Descriptions

Field	Output	Description
useLeftBound	int32*	Left bound specified inside port name argument list
useRightBound	int32	Right bound specified inside port name argument list
decLeftBound	int32	Left bound specified in port and data type declarations
decRightBound	int32	Right bound specified in port and data type declarations

The function prototypes that correspond to the field names (listed in [Table 3-36](#)) for the HierPortVectorInfo database record are as follows:

```
int MWXDb_Get_HierPortVectorInfo_useLeftBound (MWXCellId_t cellId,  
        MWXObjId_t objId, int32 *value);  
  
int MWXDb_Get_HierPortVectorInfo_useRightBound (MWXCellId_t cellId,  
        MWXObjId_t objId, int32 *value);  
  
int MWXDb_Get_HierPortVectorInfo_decLeftBound (MWXCellId_t cellId,  
        MWXObjId_t objId, int32 *value);  
  
int MWXDb_Get_HierPortVectorInfo_decRightBound (MWXCellId_t cellId,  
        MWXObjId_t objId, int32 *value);  
  
int MWXDb_Set_HierPortVectorInfo_useLeftBound (MWXCellId_t cellId,  
        MWXObjId_t objId, int32 value);  
  
int MWXDb_Set_HierPortVectorInfo_useRightBound (MWXCellId_t cellId,  
M      WXObjId_t objId, int32 value);  
  
int MWXDb_Set_HierPortVectorInfo_decLeftBound (MWXCellId_t cellId,  
        MWXObjId_t objId, int32 value);  
  
int MWXDb_Set_HierPortVectorInfo_decRightBound (MWXCellId_t cellId,  
        MWXObjId_t objId, int32 value);
```

HorizontalWire Object

The field descriptions for the HorizontalWire object are listed in [Table 3-37](#).

Table 3-37 HorizontalWire Object Field Descriptions

Field	Output	Description
netId	MWXObjId_t	Database identifier of the net associated with the wire.
wireMasterIdx	int32	Index value of associated WireMaster object. (See index field in Table 3-89 on page 3-102).
layer	int32	Number of the layer on which the wire resides. (Layer numbers appear in the technology file.)
regionNumber	int32	Zero if the wire is not associated with a region, nonzero and the regionNumber represents the identifier of the region with which the wire is associated.
startPoint	MWXPoint_t*	Specification of the x/y-coordinates of the starting point of the wire.
length	int32	Length in database units of the wire; a negative number means that the wire extends to the left of the startPoint, whereas a positive number means the wire extends to the right.
routeType	int32	See “routeType” on page 3-108 .

The prototypes for the HorizontalWire accessor functions are

```
int MWXDb_Get_HorizontalWire_netId (MWXCellId_t cellId,  
                                     MWXObjId_t objId, MWXObjId_t *nObjId);  
  
int MWXDb_Get_HorizontalWire_wireMasterIdx (MWXCellId_t cellId,  
                                             MWXObjId_t objId, int32 *value);  
  
int MWXDb_Get_HorizontalWire_layer (MWXCellId_t cellId,  
                                    MWXObjId_t objId, int32 *value);  
  
int MWXDb_Get_HorizontalWire_regionNumber (MWXCellId_t cellId,  
                                            MWXObjId_t objId, int32 *value);  
  
int MWXDb_Get_HorizontalWire_startPoint (MWXCellId_t cellId,
```

```

        MWXObjId_t objId, MWXPoint_t **point);

int MWXDb_Get_HorizontalWire_length (MWXCellId_t cellId,
        MWXObjId_t objId, int32 *value);

int MWXDb_Get_HorizontalWire_routeType(MWXCellId_t CellId,
        MWXObjId_t objId, int32 routeType);

```

See also

- [MWXDb_Set_HorizontalWire_routeType](#)

IGRPath Object

The field description for the IGRPath object is listed in [Table 3-38](#).

Table 3-38 IGRPath Field Description

Field	Output	Description
flags	int32	Routing flags with bit 1 for net class path

The prototype for the IGRPath accessor function is

```

int MWXDb_Get_IGRPath_flags (MWXCellId_t cellId,
        MWXObjId_t objId, uint32 *value));

```

InstPower Object

The field description for the InstPower object is listed in [Table 3-39](#).

Table 3-39 InstPower Field Description

Field	Output	Description
instId	int32	Instance ID

The prototype for the InstPower accessor function is

```

int MWXDb_Get_InstPower_instId (MWXCellId_t cellId,
        MWXObjId_t objId, MWXObjId_t *instId);

```

LEdge Object

The field descriptions for the LEdge object are listed in [Table 3-40](#).

Table 3-40 LEdge Field Descriptions

Field	Output	Description
nodeId1	uint32	Node 1 ID
nodeId2	uint32	Node 2 ID
nomInd	float32	Nominal inductance

The prototypes for the LEdge accessor functions are

```
int MWXDb_Get_LEdge_nodeId1(MWXCellId_t cellId,  
                             MWXObjId_t LEdgeId, uint32 *nodeId1);  
  
int MWXDb_Get_LEdge_nodeId2(MWXCellId_t cellId,  
                             MWXObjId_t LEdgeId, uint32 *nodeId2);  
  
int MWXDb_Get_LEdge_nomInd(MWXCellId_t cellId,  
                             MWXObjId_t LEdgeId, float *value);  
  
int MWXDb_Set_LEdge_nodeId1(MWXCellId_t cellId,  
                             MWXObjId_t objId, uint32 value);  
  
int MWXDb_Set_LEdge_nodeId2(MWXCellId_t cellId,  
                             MWXObjId_t objId, uint32 value);  
  
int MWXDb_Set_LEdge_nomInd(MWXCellId_t cellId,  
                             MWXObjId_t objId, float value);
```

LibInfo Object

The field description for the LibInfo object is listed in [Table 3-41](#).

Table 3-41 LibInfo Field Description

Field	Output	Description
hierDelimiter	int32	Hierarchy delimiter

The prototype for the LibInfo accessor function is

```
int MWXDb_Get_LibInfo_hierDelimiter (MWXCellId_t cellId,
                                     MWXObjId_t objId,int32 *value);
```

PreRouteSegment Object

The field description for the PreRouteSegment object is listed in [Table 3-42](#).

Table 3-42 PreRouteSegment Field Description

Field	Output	Description
dataType	int32	GDSII data type

The prototype for the PreRouteSegment accessor function is

```
int MWXDb_Get_PreRouteSegment_dataType (MWXCellId_t cellId,
                                         MWXObjId_t objId, int32 *value);
```

Net Object

The field descriptions for the Net object are listed in [Table 3-43](#).

Table 3-43 Net Object Field Descriptions

Field	Output	Description
name	char*	Hierarchical net name
type	uint32	Bit value representing the type of net (signal, power, ground, or clock); bit-and the value with one of the following constants: dbbSignalNet, dbbPowerNet, dbbGroundNet, or dbbClockNet to check the net type
numberOfPort	uint32	Number of cell ports attached to net (ports of the cell within which the net resides; for example, the net and ports are owned by the same cell)
numberOfPortInst	uint32	Number of port instances attached to the net (ports in cells instantiated within the cell that owns the net)

Table 3-43 Net Object Field Descriptions (Continued)

Field	Output	Description
cellRegionNumber	int32	Zero if the cell is not associated with a region; if nonzero, the cell region number represents the identifier (1–255) of the region with which the net is associated
numberOfChildNet	uint32	Number of nets within cell instances that are attached to the net (for example, the number of hierarchical nets)
parentId	MWXObjId_t	Database identifier of the parent net (one level of hierarchy above)
constraints	int32	Constraints for NoTouch
propertyId	int32	Property ID
index	int32	Index managed by database

The prototypes for the Net accessor functions are

```
int MWXDb_Get_Net_name (MWXCellId_t cellId,
    MWXObjId_t objId, char **string);

int MWXDb_Get_Net_type (MWXCellId_t cellId,
    MWXObjId_t objId, uint32 *value);

int MWXDb_Get_Net_numberOfPort (MWXCellId_t cellId,
    MWXObjId_t objId, uint32 *value);

int MWXDb_Get_Net_numberOfPortInst (MWXCellId_t cellId,
    MWXObjId_t objId, uint32 *value);

int MWXDb_Get_Net_cellRegionNumber (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_Net_numberOfChildNet (MWXCellId_t cellId,
    MWXObjId_t objId, uint32 *value);

int MWXDb_Get_Net_parentId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *nObjId);

int MWXDb_Get_Net_constraints (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *numOfElm, int32 **value);

int MWXDb_Get_Net_propertyId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *propertyId);
```

```
int MWXDb_Get_Net_index (MWXCellId_t cellId,  
                        MWXObjId_t objId, int32 *index);
```

Following are functions that return the object ID of the net related to that object:

- MWXDb_Get_Boundary_netId (see [Boundary Object](#))
- MWXDb_Get_Boundary_polygon_netId (see [Boundary Object](#))
- MWXDb_Get_Contact_netId (see [Contact Object](#))
- MWXDb_Get_ContactArray_netId (see [ContactArray Object](#))
- MWXDb_Get_ErrorComponent_netId (see [ErrorComponent Object](#))
- MWXDb_Get_GLink_netId (see [Glink Object](#))
- MWXDb_Get_HorizontalWire_netId (see [HorizontalWire Object](#))
- MWXDb_Get_Path_netId (see [Path Object](#))
- MWXDb_Get_PinInstMaster_childNetId (see [PinInstMaster Object](#))
- MWXDb_Get_Pin_netId (see [Pin Object](#))
- MWXDb_Get_Polygon_netId (see [Polygon Object](#))
- MWXDb_Get_PortInstMaster_netId (see [PortInstMaster Object](#))
- MWXDb_Get_PortInst_netId (see [PortInst Object](#))
- MWXDb_Get_Port_netId (see [Port Object](#))
- MWXDb_Get_Rectangle_netId (see [Rectangle Object](#))
- MWXDb_Get_VerticalWire_netId (see [VerticalWire Object](#))

Additional functions related to Net are

- [MWXDb_Add_NetSubType](#)
- [MWXDb_Add_NetSubType_ByName](#)
- [MWXDb_Connect](#)
- [MWXDb_Connect_PortInstance_ByName](#)
- [MWXDb_Connect_PortWithPin](#)
- [MWXDb_Create_ErrorComponents](#)
- [MWXDb_Create_Glink](#)

- [MWXDb_Create_Net](#)
- [MWXDb_Create_Pin](#)
- [MWXDb_Create_PinWithPortName](#)
- [MWXDb_Disconnect](#)
- [MWXDb_Disconnect_PortInstance_ByName](#)
- [MWXDb_Display_DiodePortInst_FromNet](#)
- [MWXDb_Get_ContactGeom](#)
- [MWXDb_Get_DesignRuleDiffNetMinSpacing](#)
- [MWXDb_Get_GlobalNet_ByNameMap](#)
- [MWXDb_Get_LayerLayoutAttr](#)
- [MWXDb_Get_NetComponents](#)
- [MWXDb_Get_NetLoad](#)
- [MWXDb_Get_NetPort](#)
- [MWXDb_Get_NetPortInst](#)
- [MWXDb_Get_NetSwitchActivity](#)
- [MWXDb_Get_NetWiring](#)
- [MWXDb_Get_Net_ByName](#)
- [MWXDb_Is_NetSubTypeCoreAndPad](#)
- [MWXDb_Is_NetSubTypeCoreOnly](#)
- [MWXDb_Is_NetSubTypeCTSNet](#)
- [MWXDb_Is_NetSubTypeCTSRootNet](#)
- [MWXDb_Is_NetSubTypeNonCTSNet](#)
- [MWXDb_Is_NetSubTypePadOnly](#)
- [MWXDb_Set_Netload](#)
- [MWXDb_Set_NetSubType](#)
- [MWXDb_Set_NetSubType_ByName](#)
- [MWXDb_Set_NetSwitchActivity](#)
- [MWXDb_Set_NetType](#)

- [MWXDb_Set_NetType_ByName](#)
- [MWXDb_Purge_AllSingleNet](#)
- [MWXDb_Purge_Net](#)

Path Object

The field descriptions for the Path object are listed in [Table 3-44](#).

Table 3-44 Path Object Field Descriptions

Field	Output	Description
bbox	4 of int32	Two x/y-coordinate pairs representing the lower-left and upper-right corners of the bounding box of the path.
netId	MWXObjId_t	Database identifier of the net associated with the path.
pathType	int32	0 (bit 0 is set) if the path edge aligns with the coordinate, 2 (bit 1 is set) if the path edge extends a half-width beyond the coordinate; bit-and with the constant dbcSquareEnds or dbcSquareEndsByHalfWidth to check the value.
layer	int32	Number of the layer on which the path resides. (Layer numbers appear in the technology file.)
width	uint32	Width of the path, in database units.
numberOfPoint	uint32	Number of points used to define the path.
points	uint32, MWXPoint_t array	Number of points used to define the path and an array of the x/y-coordinate pairs.

The prototypes for the Path accessor functions are

```
int MWXDb_Get_Path_bbox (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *left, int32 *bottom, int32 *right, int32 *top);

int MWXDb_Get_Path_netId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *nObjId);
```



```

int MWXDb_Get_Path_pathType (MWXCellId_t cellId,
                             MWXObjId_t objId, int32 *value);

int MWXDb_Get_Path_layer (MWXCellId_t cellId,
                          MWXObjId_t objId, int32 *value);

int MWXDb_Get_Path_width (MWXCellId_t cellId,
                          MWXObjId_t objId, uint32 *value);

int MWXDb_Get_Path_numberOfPoint (MWXCellId_t cellId,
                                  MWXObjId_t objId, uint32 *value);

int MWXDb_Get_Path_points (MWXCellId_t cellId,
                          MWXObjId_t objId, int32 *numOfPoint, MWXPoint_t **points);

```

Additional functions related to Path are

- MWXDb_Get_WireMaster_pathType (see [WireMaster Object](#))
- [MWXDb_Convert_ObjectsToPath](#)
- [MWXDb_Convert_PathToPolygon](#)
- [MWXDb_Convert_PathToRectangles](#)
- [MWXDb_Create_Path](#)
- [MWXDb_Create_Wire_Master](#)

Pin Object

The field descriptions for the Pin object are listed in [Table 3-45](#).

Table 3-45 Pin Object Field Descriptions

Field	Output	Description
name	char*	Name of the pin.
portId	MWXObjId_t	Database identifier of related port.
accessDirection	int32	See “ accessDirection ” on page 3-106 .
statusFlag	int32	See “ statusFlag ” on page 3-109 .

Table 3-45 Pin Object Field Descriptions (Continued)

Field	Output	Description
statusFlag1	int32	Bit 0 (masked with constant dbbPinFollowInRow) set to 1 indicates that the pin should follow the row; bit 1 (masked with constant dbbPinFollowInCol) set to 1 indicates that the pin should follow the column.
netId	MWXObjId_t	Database identifier of net within the owning cell to which the pin is attached.
electricalEquivalentClass	uint32	A byte-field in which the first 2 bytes (bits 0–15) define electrically equivalent pins (maximum of 65,535 sets of equivalent pins) and the second 2 bytes (bits 16–31) represent must-join pins (the router externally connects all pins on the same port with the same must-join value without regard to order); the default value for byte pairs is 0.
bbox	4 of int32	Two x/y-coordinate pairs representing the lower-left and upper-right corners of the bounding box of the pin.
layer	int32	Number of the layer on which the pin resides. (Layer numbers appear in the technology file.)
numberOfPoint	int32	Number of points used to define the pin.

Table 3-45 Pin Object Field Descriptions (Continued)

Field	Output	Description
points	uint32, MWXPoint_t array	Number of points used to define the pin and an array of the x/y-coordinate pairs. Note: If a negative number is returned, use the following formula to determine the number of rectangles: $(-numberOfPoint)/2$ rectangles. For example, if the return value is -14, the number of rectangles is 7.
dataType	int32	GDSII data type.
firstViaSolution	int32	List of via solution.
firstWireSolution	int32	List of wire solution.

The prototypes for the Pin accessor functions are

```

int MWXDb_Get_Pin_name (MWXCellId_t cellId,
    MWXObjId_t objId, char **string);

int MWXDb_Get_Pin_portId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *nObjId);

int MWXDb_Get_Pin_accessDirection (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_Pin_statusFlag (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_Pin_statusFlag1 (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_Pin_netId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *nObjId);

int MWXDb_Get_Pin_electricalEquivalentClass (MWXCellId_t cellId,
    MWXObjId_t objId, uint32 *value);

int MWXDb_Get_Pin_bbox (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *left, int32 *bottom, int32 *right, int32 *top);

int MWXDb_Get_Pin_layer (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_Pin_numberOfPoint (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

```

```

int MWXDb_Get_Pin_points (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *numOfPoint, MWXPoint_t **points);

int MWXDb_Get_Pin_dataType (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_Pin_firstViaSolutionId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *firstViaSolution);

int MWXDb_Get_Pin_firstWireSolutionId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *firstWireSolution);

```

Additional functions related to Pin are

- MWXDb_Get_CellInstMaster_numberOfPin (see [CellInstMaster Object](#))
- MWXDb_Get_PinAccessEdge_pinId (see [PinAccessEdge Object](#))
- MWXDb_Get_Port_numberOfPin (see [Port Object](#))
- [MWXDb_Clear_CellMustJoinPin](#)
- [MWXDb_Connect_PortWithPin](#)
- [MWXDb_Create_Pin](#)
- [MWXDb_Create_PinWithPortName](#)
- [MWXDb_Delete_Pin](#)
- [MWXDb_Get_AccessEdgeFromPin](#)
- [MWXDb_Get_AllPin_ByName](#)
- [MWXDb_Get_Pin_ByName](#)
- [MWXDb_Get_PinFromPort](#)
- [MWXDb_Set_CellMustJoin2Pin](#)
- [MWXDb_Set_CellMustJoinPinPattern](#)

PinAccessEdge Object

The field descriptions for the PinAccessEdge object are listed in [Table 3-46](#).

Table 3-46 PinAccessEdge Object Field Descriptions

Field	Output	Description
pinId	MWXObjId_t	Database identifier of related pin.
accessDir	int32	See “ accessDirection ” on page 3-106 .
layer	int32	Number of the layer on which the PinAccessEdge object resides.
width	int32	Width of the access edge.
center	MWXPoint_t	Center point of the PinAccessEdge object.
bbox	4 of int32	Two x/y-coordinate pairs representing the lower-left and upper-right corners of the bounding box of the pin.

The prototypes for the PinAccessEdge accessor functions are

```
int MWXDb_Get_PinAccessEdge_pinId ((MWXCellId_t cellId,  
    MWXObjId_t objId, MWXObjId_t *nObjId));  
  
int MWXDb_Get_PinAccessEdge_accessDir ((MWXCellId_t cellId,  
    MWXObjId_t objId, int32 *value));  
  
int MWXDb_Get_PinAccessEdge_layer ((MWXCellId_t cellId,  
    MWXObjId_t objId, int32 *value));  
  
int MWXDb_Get_PinAccessEdge_width ((MWXCellId_t cellId,  
    MWXObjId_t objId, int32 *value));  
  
int MWXDb_Get_PinAccessEdge_center ((MWXCellId_t cellId,  
    MWXObjId_t objId, MWXPoint_t **point));  
  
int MWXDb_Get_PinAccessEdge_bbox ((MWXCellId_t cellId,  
    MWXObjId_t objId, int32 *left, int32 *bottom, int32 *right, int32 *top));
```

PinInstMaster Object

The field descriptions for the PinInstMaster object are listed in [Table 3-47](#).

Table 3-47 PinInstMaster Object Field Descriptions

Field	Output	Description
name	char*	Name of the related pin.
cellInstMasterId	MWXObjId_t	Database identifier of the related cell instance.
childPinId	MWXObjId_t	Database identifier of the pin inside the cell related to the cellInstMasterId object.
childNetId	MWXObjId_t	Database identifier of the net inside the cell related to the cellInstMasterId object.
childPortId	MWXObjId_t	Database identifier of the port inside the cell related to the cellInstMasterId object.
accessDirection	int32	See “accessDirection” on page 3-106 .
statusFlag	int32	See “statusFlag” on page 3-109 .
statusFlag1	int32	Bit 0 (masked with constant dbbPinFollowInRow) set to 1 indicates that the pin should follow the row; bit 1 (masked with constant dbbPinFollowInCol) set to 1 indicates that the pin should follow the column.

Table 3-47 PinInstMaster Object Field Descriptions (Continued)

Field	Output	Description
electricalEquivalentClass	uint32	A byte field in which the first 2 bytes (bits 0–15) define electrically equivalent pins (maximum of 65,535 sets of equivalent pins) and the second 2 bytes (bits 16–31) represent must-join pins (the router externally connects all pins on the same port with the same must-join value without regard to order); the default value for byte pairs is 0.
bbox	4 of int32	Two x/y-coordinate pairs representing the lower-left and upper-right corners of the bounding box of the pin.
layer	int32	Number of the layer on which the pin instances associated with the PinInstMaster object reside. (Layer numbers appear in the technology file.)
numberOfPoint	int32	Number of points used to define the pin instances associated with the PinInstMaster object.
points	uint32, MWXPoint_t array	Number of points used to define the pin instances associated with the PinInstMaster object and an array of the x/y-coordinate pairs.

The prototypes for the PinInstMaster accessor functions are

```
int MWXDb_Get_PinInstMaster_name (MWXCellId_t cellId,
    MWXObjId_t objId, char **string);

int MWXDb_Get_PinInstMaster_cellInstMasterId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *nObjId);

int MWXDb_Get_PinInstMaster_childPinId (MWXCellId_t cellId,
```

```

        MWXObjId_t objId, MWXObjId_t *nObjId);

int MWXDb_Get_PinInstMaster_childNetId (MWXCellId_t cellId,
        MWXObjId_t objId, MWXObjId_t *nObjId);

int MWXDb_Get_PinInstMaster_childPortId (MWXCellId_t cellId,
        MWXObjId_t objId, MWXObjId_t *nObjId);

int MWXDb_Get_PinInstMaster_accessDirection (MWXCellId_t cellId,
        MWXObjId_t objId, int32 *value);

int MWXDb_Get_PinInstMaster_statusFlag (MWXCellId_t cellId,
        MWXObjId_t objId, int32 *value);

int MWXDb_Get_PinInstMaster_statusFlag1 (MWXCellId_t cellId,
        MWXObjId_t objId, int32 *value);

int MWXDb_Get_PinInstMaster_electricalEquivalentClass (MWXCellId_t cellId,
        MWXObjId_t objId, uint32 *value);

int MWXDb_Get_PinInstMaster_bbox (MWXCellId_t cellId,
        MWXObjId_t objId, int32 *left, int32 *bottom, int32 *right, int32 *top);

int MWXDb_Get_PinInstMaster_layer (MWXCellId_t cellId,
        MWXObjId_t objId, int32 *value);

int MWXDb_Get_PinInstMaster_numberOfPoint (MWXCellId_t cellId,
        MWXObjId_t objId, int32 *value);

int MWXDb_Get_PinInstMaster_points (MWXCellId_t cellId,
        MWXObjId_t objId, int32 *numOfPoint, MWXPoint_t **points);

```

See also

- [MWXDb_Display_DiodePinInstMaster_FromDiodePortInstMaster](#)

PlanGroup Object

The field descriptions for the PlanGroup object are listed in [Table 3-48](#).

Table 3-48 PlanGroup Object Field Descriptions

Field	Output	Description
name	char*	Name of the planGroup object
bbox	4 of int32	Two x/y-coordinate pairs representing the lower-left and upper-right corners of the bounding box of the array
type	int32	The plan group's type

Table 3-48 PlanGroup Object Field Descriptions

Field	Output	Description
cellInstGroupId	MWXObjId_t	The database identifier of the cellInstGroup object
planStatus	uint32	The current status of the planGroup object
numberOfCellRow	uint32	The number of cellRow objects in the plan

The prototypes for the PlanGroup accessor functions are

```
int MWXDb_Get_PlanGroup_name (MWXCellId_t cellId,
                             MWXObjId_t objId, MWXObjId_t, char**name);

int MWXDb_Get_PlanGroup_bbox (MWXCellId_t cellId,
                             MWXObjId_t objId, int32*left, int32 *bottom, int32 *right, int32 *top);

int MWXDb_Get_PlanGroup_type (MWXCellId_t cellId,
                             MWXObjId_t objId, int32 *type);

int MWXDb_Get_PlanGroup_cellInstGroupId (MWXCellId_t cellId,
                                         MWXObjId_t objId, MWXObjId_t *cellInstGroupId);

int MWXDb_Get_PlanGroup_planStatus (MWXCellId_t cellId,
                                     MWXObjId_t objId, uint32* planStatus);

int MWXDb_Get_PlanGroup_numberOfCellRow (MWXCellId_t cellId,
                                         MWXObjId_t objId, unit32 *numberOfCellRow);
```

Additional functions related to PlanGroup are

- [MWXDb_Add_CellInst_ToPlanGroup_ByName](#)
- [MWXDb_Create_PlanGroup](#)
- [MWXDb_Get_PlanGroup_ByName](#)
- [MWXDb_Purge_AllPlanGroup](#)

Polygon Object

The field descriptions for the Polygon object are listed in [Table 3-49](#).

Table 3-49 Polygon Object Field Descriptions

Field	Output	Description
bbox	4 of int32	Two x/y-coordinate pairs representing the lower-left and upper-right corners of the bounding box of the polygon.
netId	MWXObjId_t	Database identifier of the net to associated with the polygon.
layer	int32	Number of the layer on which the polygon resides. (Layer numbers appear in the technology file.)
numberOfPoint	uint32	Number of points used to define the polygon.
points	uint32, MWXPoint_t array	Number of points used to define the polygon and an array of the x/y-coordinate pairs.

The prototypes for the Polygon accessor functions are

```
int MWXDb_Get_Polygon_bbox (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *left, int32 *bottom, int32 *right, int32 *top);

int MWXDb_Get_Polygon_netId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *nObjId);

int MWXDb_Get_Polygon_layer (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_Polygon_numberOfPoint (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *numOfPoint);

int MWXDb_Get_Polygon_points (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *numOfPoint, MWXPoint_t **points);
```

Additional functions related to Polygon are

- [MWXDb_BooleanAnd_Objects](#)
- [MWXDb_BooleanNot_Objects](#)
- [MWXDb_BooleanOr_Objects](#)
- [MWXDb_BooleanXor_Objects](#)

- [MWXDb_Convert_PathToPolygon](#)
- [MWXDb_Convert_PolygonToRectangles](#)
- [MWXDb_Convert_RectangleToPolygon](#)
- [MWXDb_Create_Polygon](#)
- [MWXDb_Get_Boundary_polygon_netId](#) (see [Boundary Object](#))

Port Object

The field descriptions for the Port object are listed in [Table 3-50](#).

Table 3-50 Port Object Field Descriptions

Field	Output	Description
name	char*	Name of the port.
direction	int32	See “ direction ” on page 3-105.
numberOfPin	uint32	Number of pins associated with the port.
netId	MWXObjId_t	Database identifier of the net within the owning cell to which the port is attached.
mustJoinRootId	int32	Must-join object ID.
index	int32	Index managed by the database.
type	int32	Type of port.

The prototypes for the Port accessor functions are

```
int MWXDb_Get_Port_name (MWXCellId_t cellId,
    MWXObjId_t objId, char **string);

int MWXDb_Get_Port_direction (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_Port_numberOfPin (MWXCellId_t cellId,
    MWXObjId_t objId, uint32 *value);

int MWXDb_Get_Port_netId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *nObjId);

int MWXDb_Get_Port_mustJoinRootId (MWXCellId_t cellId,
```

```

        MWXObjId_t objId, MWXObjId_t *mustJoinRootId);

int MWXDb_Get_Port_index (MWXCellId_t cellId,
        MWXObjId_t objId, int32 *index);

int MWXDb_Get_Port_type (MWXCellId_t cellId,
        MWXObjId_t objId, int32 *type);

```

Additional functions related to Port are

- MWXDb_Get_CellInstMaster_numberOfPort (see [CellInstMaster Object](#))
- MWXDb_Get_Net_numberOfPort (see [Net Object](#))
- MWXDb_Get_PinInstMaster_childPortId (see [PortInstMaster Object](#))
- MWXDb_Get_Pin_portId ([Pin Object](#))
- MWXDb_Get_ViaRegion_portId (see [ViaRegion Object](#))
- [MWXDb_Clear_CellMustJoinPin](#)
- [MWXDb_Connect_PortWithPin](#)
- [MWXDb_Create_PinWithPortName](#)
- [MWXDb_Get_NetPort](#)
- [MWXDb_Get_PinFromPort](#)
- [MWXDb_Get_Port_ByName](#)
- [MWXDb_Get_ViaRegionFromPort](#)
- [MWXDb_Set_CellMustJoin2Pin](#)
- [MWXDb_Set_CellMustJoinPinPattern](#)
- [MWXDb_Set_TdfLoadCapacitance](#)
- [MWXDb_Set_TdfLoadCapacitanceInDouble](#)
- [MWXDb_Set_TdfSrcCapacitanceList](#)
- [MWXDb_Set_TdfSrcCapacitanceListInDouble](#)
- [MWXDb_Set_TdfSrcResistanceList](#)
- [MWXDb_Set_TdfSrcResistanceListInDouble](#)

PortCap Object

Provides six port capacitance values, as shown in [Table 3-51](#).

Table 3-51 PortCap Object Field Description

Field	Output	Description
capacitance	int32	Specifies the capacitance information of the PortCap object. The capacitance information is provided as a set of six values that include the following: riseMin - rise minimum value riseNom - rise normal value riseMax - rise maximum value fallMin - fall minimum value fallNom - fall normal value fallMax - fall maximum value

The prototypes for the PortCap accessor functions are

```
int MWXDb_Get_PortCap_capacitance(MWXCellId_t  
cellId, MWXObjId_t portCapId, int32* riseMin,  
int32* riseNom, int32* riseMax, int32* fallMin,  
int32* fallNom, int32* fallMax)
```

PortFunc Object

The field descriptions for the Portfunc object are listed in [Table 3-52](#).

Table 3-52 PortFunc Object Field Descriptions

Field	Output	Description
type	int32	Port function type with value 1 for the enable active exp type and value 2 for the data pin exp type
condStr	char*	Condition string

The prototypes for the PortFunc accessor functions are

```
int MWXDb_Get_PortFunc_type (MWXCellId_t cellId,  
MWXObjId_t objId, int32 *type);  
  
int MWXDb_Get_PortFunc_condStr (MWXCellId_t cellId,  
MWXObjId_t objId, char **string);
```

PortInst Object

The field descriptions for the PortInst object are listed in [Table 3-53](#).

Table 3-53 PortInst Object Field Descriptions

Field	Output	Description
netId	MWXObjId_t	Database identifier of the net attached to the port instance
portInstMasterId	MWXObjId_t	Database identifier of the master port definition
localStatus	int32	Bit field specifying the following: bit 0 (constant mask dbbPortInstAutoTie) is set to indicate that the port is tied to either power or ground; bit 1 (constant mask dbbPortInstTieHigh) is set to 1 if the port is tied to power, and 0 if tied to ground (used in conjunction with bit 0); and bit 2 (constant mask dbbPortInstNotChild) is set to indicate that the port is not a member of the net
cellInstId	MWXCellId_t	Database identifier of the CellInst object that owns the PortInst object

The prototypes for the PortInst accessor functions are

```
int MWXDb_Get_PortInst_netId (MWXCellId_t cellId,
                              MWXObjId_t objId, MWXObjId_t *nObjId);

int MWXDb_Get_PortInst_portInstMasterId (MWXCellId_t cellId,
                                          MWXObjId_t objId, MWXObjId_t *nObjId);

int MWXDb_Get_PortInst_localStatus (MWXCellId_t cellId,
                                    MWXObjId_t objId, int32 *value);

int MWXDb_Get_PortInst_cellInstId (MWXCellId_t cellId,
                                   MWXObjId_t objId, MWXObjId_t *nObjId);
```

Additional functions related to PortInst are

- MWXDb_Get_Net_numberOfPortInst (see [Net Object](#))
- [MWXDb_Connect_PortInstance_ByName](#)
- [MWXDb_Disconnect_PortInstance_ByName](#)
- [MWXDb_Display_DiodePortInst](#)

- [MWXDb_Display_DiodePortInst_FromCellInst](#)
- [MWXDb_Display_DiodePortInst_FromNet](#)
- [MWXDb_Get_NetPortInst](#)
- [MWXDb_Get_PortInstFromCellInst](#)
- [MWXDb_Show_DiodePortConn](#)

PortInstMaster Object

The field descriptions for the PortInstMaster object are listed in [Table 3-54](#).

Table 3-54 PortInstMaster Object Field Descriptions

Field	Output	Description
name	char*	Name of the related port.
cellInstMasterId	MWXObjId_t	Database identifier of the related CellInstMaster object.
direction	int32	See “ direction ” on page 3-105.
numberOfPinInstMaster	uint32	Number of related PinInstMaster objects.
portId	MWXObjId_t	Database identifier of the related port.
netId	MWXObjId_t	Database identifier of the net within the cell instance that is attached to the related port instances (for example, child net, one level of hierarchy below the port instances).

The prototypes for the PortInstMaster accessor functions are

```
int MWXDb_Get_PortInstMaster_name (MWXCellId_t cellId,
    MWXObjId_t objId, char **string);

int MWXDb_Get_PortInstMaster_cellInstMasterId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *nObjId);

int MWXDb_Get_PortInstMaster_direction (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);
```

```
int MWXDb_Get_PortInstMaster_numberOfPinInstMaster (MWXCellId_t cellId,
    MWXObjId_t objId, uint32 *value);

int MWXDb_Get_PortInstMaster_portId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *nObjId);

int MWXDb_Get_PortInstMaster_netId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *nObjId);
```

Additional functions related to PortInstMaster are

- MWXDb_Get_PortInst_portInstMasterId (see [PortInst Object](#))
- [MWXDb_Display_DiodePinInstMaster_FromDiodePortInstMaster](#)
- [MWXDb_Display_DiodePortInstMaster](#)
- [MWXDb_Display_DiodePortInstMaster_FromDiodeCellInstMaster](#)

PortToPortDelay Object

The field descriptions for the PortToPortDelay object are listed in [Table 3-55](#).

Table 3-55 PortToPortDelay Object Field Descriptions

Field	Output	Description
recType	int32	See “ recType ” on page 3-108.
startCond	char*	Start condition string.
endCond	char*	End condition string.

The prototypes for the PortToPortDelay accessor functions are

```
int MWXDb_Get_PortToPortDelay_recType (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_PortToPortDelay_startCond (MWXCellId_t cellId,
    MWXObjId_t objId, char **string);

int MWXDb_Get_PortToPortDelay_endCond (MWXCellId_t cellId,
    MWXObjId_t objId, char **string));
```

PortToPortTable Object

The field descriptions for the PortToPortTable object are listed in [Table 3-56](#).

Table 3-56 PortToPortTable Object Field Descriptions

Field	Output	Description
unateness	int32	Unateness with 0 for noninvert, 1 for invert, and 2 for nonunate.
timingType	int32	Timing type.
status	int32	Delay status with 0 for no nominal delay, and 1 otherwise for normal delay.
recType	int32	See “ recType ” on page 3-108.
fromPortId	MWXObjId_t	From port ID.
toPortId	MWXObjId_t	To port ID.
startCond	char*	Start condition string ID.
minRisePropDelayId	MWXObjId_t	Minimum rise propagation delay ID.
maxRisePropDelayId	MWXObjId_t	Maximum rise propagation delay ID.
minFallPropDelayId	MWXObjId_t	Minimum fall propagation delay ID.
maxFallPropDelayId	MWXObjId_t	Maximum fall propagation delay ID.
minRiseTranDelayId	MWXObjId_t	Minimum rise transition delay ID.
maxRiseTranDelayId	MWXObjId_t	Maximum rise transition delay ID.
minFallTranDelayId	MWXObjId_t	Minimum fall transition delay ID.
maxFallTranDelayId	MWXObjId_t	Maximum fall transition delay ID.
nomFallPropDelayId	MWXObjId_t	Nominal rise propagation delay ID.
nomRiseTranDelayId	MWXObjId_t	Nominal rise transition delay ID.
nomFallTranDelayId	MWXObjId_t	Nominal fall transition delay ID.

The prototypes for the PortToPortTable accessor functions are

```

int MWXDb_Get_PortToPortTable_unateness (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *unateness);

int MWXDb_Get_PortToPortTable_timingType (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *timingType);

int MWXDb_Get_PortToPortTable_status (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *status);

int MWXDb_Get_PortToPortTable_recType (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *recType);

int MWXDb_Get_PortToPortTable_fromPortId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *fromPortId);

int MWXDb_Get_PortToPortTable_toPortId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *toPortId);

int MWXDb_Get_PortToPortTable_startCond (MWXCellId_t cellId,
    MWXObjId_t objId, char **string);

int MWXDb_Get_PortToPortTable_minRisePropDelayId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *minRisePropDelayId);

int MWXDb_Get_PortToPortTable_maxRisePropDelayId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *maxRisePropDelayId);

int MWXDb_Get_PortToPortTable_minFallPropDelayId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *minFallPropDelayId);

int MWXDb_Get_PortToPortTable_maxFallPropDelayId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *maxFallPropDelayId);

int MWXDb_Get_PortToPortTable_minRiseTranDelayId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *minRiseTranDelayId);

int MWXDb_Get_PortToPortTable_maxRiseTranDelayId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *maxRiseTranDelayId);

int MWXDb_Get_PortToPortTable_minFallTranDelayId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *minFallTranDelayId);

int MWXDb_Get_PortToPortTable_maxFallTranDelayId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *maxFallTranDelayId);

int MWXDb_Get_PortToPortTable_nomFallPropDelayId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *nomFallPropDelayId);

int MWXDb_Get_PortToPortTable_nomRiseTranDelayId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *nomRiseTranDelayId);

int MWXDb_Get_PortToPortTable_nomFallTranDelayId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *nomFallTranDelayId);

```

Rectangle Object

The field descriptions for the Rectangle object are listed in [Table 3-57](#).

Table 3-57 Rectangle Object Field Descriptions

Field	Output	Description
netId	MWXObjId_t	Database identifier of the net to which object is attached.
layer	int32	Number of the layer on which the rectangle resides. (Layer numbers appear in the technology file.)
bbox	4 of int32	Two x/y-coordinate pairs representing the lower-left and upper-right corners of the bounding box of the rectangle.

The prototypes for the Rectangle accessor functions are

```
int MWXDb_Get_Rectangle_netId (MWXCellId_t cellId,  
                               MWXObjId_t objId, MWXObjId_t *nObjId);  
  
int MWXDb_Get_Rectangle_layer (MWXCellId_t cellId,  
                               MWXObjId_t objId, int32 *value);  
  
int MWXDb_Get_Rectangle_bbox (MWXCellId_t cellId,  
                              MWXObjId_t objId, int32 *left, int32 *bottom, int32 *right, int32 *top);
```

Additional functions related to Rectangle are

- [MWXDb_Convert_PathToRectangles](#)
- [MWXDb_Convert_PolygonToRectangles](#)
- [MWXDb_Convert_RectangleToPolygon](#)
- [MWXDb_Create_Rectangle](#)

RectArray Object

The field description for the RectArray object is listed in [Table 3-58](#).

Table 3-58 RectArray Object Field Description

Field	Output	Description
dataType	int32	GDSII data type

The prototype for the RectArray accessor function is

```
int MWXDb_Get_RectArray_dataType P_((MWXCellId_t cellId,  
    MWXObjId_t objId, int32 *value));
```

REdge Object

The field descriptions for the REdge object are listed in [Table 3-59](#).

Table 3-59 REdge Field Descriptions

Field	Output	Description
nodeId1	uint32	Node 1 ID
nodeId2	uint32	Node 2 ID
nomRes	float32	Nominal resistance

The prototypes for the REdge accessor functions are

```
int MWXDb_Get_REdge_nodeId1(MWXCellId_t cellId,  
    MWXObjId_t REdgeId, uint32 *value);  
  
int MWXDb_Get_REdge_nodeId2(MWXCellId_t cellId,  
    MWXObjId_t REdgeId, uint32 *value);  
  
int MWXDb_Get_REdge_nomRes(MWXCellId_t cellId,  
    MWXObjId_t REdgeId, float *value);  
  
int MWXDb_Set_REdge_nodeId1(MWXCellId_t cellId,  
    MWXObjId_t objId, uint32 value);  
  
int MWXDb_Set_REdge_nodeId2(MWXCellId_t cellId,  
    MWXObjId_t objId, uint32 value);  
  
int MWXDb_Set_REdge_nomRes(MWXCellId_t cellId,
```

```
MWXObjId_t objId, float value);
```

RegionBlockage Object

The field descriptions for the RegionBlockage object are listed in [Table 3-60](#).

Table 3-60 RegionBlockage Field Descriptions

Field	Output	Description
xAnchor	int32	Horizontal anchor, left = 0 and right = 1
xOffset	int32	Horizontal offset
yAnchor	int32	Vertical anchor, bottom = 0 and top = 1
yOffset	int32	Vertical offset

The prototypes for the RegionBlockage accessor functions are

```
int MWXDb_Get_RegionBlockage_xAnchor (MWXCellId_t cellId,  
    MWXObjId_t objId, int32 *value);  
  
int MWXDb_Get_RegionBlockage_xOffset (MWXCellId_t cellId,  
    MWXObjId_t objId, int32 *value);  
  
int MWXDb_Get_RegionBlockage_yAnchor (MWXCellId_t cellId,  
    MWXObjId_t objId, int32 *value);  
  
int MWXDb_Get_RegionBlockage_yOffset (MWXCellId_t cellId,  
    MWXObjId_t objId, int32 *value);
```

RLCCoupleNet Object

The field descriptions for the RLCCoupleNet object are listed in [Table 3-61](#).

Table 3-61 RLCCoupleNet Field Descriptions

Field	Output	Description
flag	unsigned char	Flag
filler	unsigned char	Filler
rlcNetId1	int32	RLC net 1 ID

Table 3-61 *RLCCoupleNet Field Descriptions*

Field	Output	Description
rlcNetId2	int32	RLC net 2 ID
cedgeAttachId	int32	Attachment for capacitance edge
ledgeAttachId	int32	Attachment for inductance edge

The prototypes for the RLCCoupleNet accessor functions are

```
int MWXDb_Get_RLCCoupleNet_flag(MWXCellId_t cellId,
                                MWXObjId_t RLCCoupleNetId, int32 *value);

int MWXDb_Get_RLCCoupleNet_filler(MWXCellId_t cellId,
                                MWXObjId_t RLCCoupleNetId, int32 *numberOfElm, int32 **filler);

int MWXDb_Get_RLCCoupleNet_rlcNetId1(MWXCellId_t cellId,
                                MWXObjId_t RLCCoupleNetId, MWXObjId_t *rlcNetId1);

int MWXDb_Get_RLCCoupleNet_rlcNetId2(MWXCellId_t cellId,
                                MWXObjId_t RLCCoupleNetId, MWXObjId_t *rlcNetId2);

int MWXDb_Get_RLCCoupleNet_cedgeAttachId(MWXCellId_t cellId,
                                MWXObjId_t RLCCoupleNetId, MWXObjId_t *value);

int MWXDb_Get_RLCCoupleNet_ledgeAttachId(MWXCellId_t cellId,
                                MWXObjId_t RLCCoupleNetId, MWXObjId_t *value);

int MWXDb_Set_RLCCoupleNet_flag(MWXCellId_t cellId,
                                MWXObjId_t objId, int32 value);

int MWXDb_Set_RLCCoupleNet_filler(MWXCellId_t cellId,
                                MWXObjId_t objId, int32 elm, int32 *value);

int MWXDb_Set_RLCCoupleNet_rlcNetId1(MWXCellId_t cellId,
                                MWXObjId_t objId, MWXObjId_t nObjId);

int MWXDb_Set_RLCCoupleNet_rlcNetId2(MWXCellId_t cellId,
                                MWXObjId_t objId, MWXObjId_t nObjId);

int MWXDb_Set_RLCCoupleNet_cedgeAttachId(MWXCellId_t cellId,
                                MWXObjId_t objId, MWXObjId_t nObjId);

int MWXDb_Set_RLCCoupleNet_ledgeAttachId(MWXCellId_t cellId,
                                MWXObjId_t objId, MWXObjId_t nObjId);
```

RLCNet Object

The field descriptions for the RLCNet object are listed in [Table 3-62](#).

Table 3-62 RLCNet Field Descriptions

Field	Output	Description
netNameId	uint32	HN code for nets extracted by the Star-RC tool or net name ID for nets extracted by Apollo-RC tool
source	unsigned char	Generator of the net
flag	unsigned char	Flag: bits 4–7 routing status
flag2	unsigned char	Flag 2: when used by AstroRail extraction, first 2 bits are used to identify usage of net's extraction, and next 3 bits are used for net type
filler	unsigned char	Filler
portAttachId	int32	Linked list of port instance
nodeAttachId	int32	Linked list of RLC nodes
redgeAttachId	int32	Linked list of R edges
ledgeAttachId	int32	Linked list of L edges
coupleNetAttachId	int32	Linked list of coupled nets

The prototypes for the RLCNet accessor functions are

```
int MWXDb_Get_RLCNet_netNameId(MWXCellId_t cellId,  
                                MWXObjId_t RLCNetId, uint32 *netNameId);  
  
int MWXDb_Get_RLCNet_source(MWXCellId_t cellId,  
                             MWXObjId_t RLCNetId, int32 *value);  
  
int MWXDb_Get_RLCNet_flag(MWXCellId_t cellId,  
                           MWXObjId_t RLCNetId, int32 *value);  
  
int MWXDb_Get_RLCNet_flag2(MWXCellId_t cellId,  
                            MWXObjId_t RLCNetId, int32 *value);  
  
int MWXDb_Get_RLCNet_portAttachId(MWXCellId_t cellId,  
                                   MWXObjId_t RLCNetId, MWXObjId_t *value);
```

```

int MWXDb_Get_RLCNet_nodeAttachId(MWXCellId_t cellId,
    MWXObjId_t RLCNetId, MWXObjId_t *value);

int MWXDb_Get_RLCNet_redgeAttachId(MWXCellId_t cellId,
    MWXObjId_t RLCNetId, MWXObjId_t *value);

int MWXDb_Get_RLCNet_ledgeAttachId(MWXCellId_t cellId,
    MWXObjId_t RLCNetId, MWXObjId_t *value);

int MWXDb_Get_RLCNet_filler(MWXCellId_t cellId,
    MWXObjId_t RLCNetId, int32 *value);

int MWXDb_Get_RLCNet_coupleNetAttachId(MWXCellId_t cellId,
    MWXObjId_t RLCNetId, MWXObjId_t *value);

int MWXDb_Set_RLCNet_netNameId(MWXCellId_t cellId,
    MWXObjId_t objId, uint32 value);

int MWXDb_Set_RLCNet_source(MWXCellId_t cellId,
    MWXObjId_t objId, int32 value);

int MWXDb_Set_RLCNet_flag(MWXCellId_t cellId,
    MWXObjId_t objId, int32 value);

int MWXDb_Set_RLCNet_flag2(MWXCellId_t cellId,
    MWXObjId_t objId, int32 value);

int MWXDb_Set_RLCNet_filler(MWXCellId_t cellId,
    MWXObjId_t objId, int32 value);

int MWXDb_Set_RLCNet_portAttachId(MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t nObjId);

int MWXDb_Set_RLCNet_nodeAttachId(MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t nObjId);

int MWXDb_Set_RLCNet_redgeAttachId(MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t nObjId);

int MWXDb_Set_RLCNet_ledgeAttachId(MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t nObjId);

int MWXDb_Set_RLCNet_coupleNetAttachId(MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t nObjId);

```

RLCNode Object

The field descriptions for the RLCNode object are listed in [Table 3-63](#).

Table 3-63 RLCNode Field Descriptions

Field	Output	Description
nodeId	uint32	Node ID
nomCap	float32	Nominal capacitance

The prototypes for the RLCNode accessor functions are

```
int MWXDb_Get_RLCNode_nodeId(MWXCellId_t cellId,  
                              MWXObjId_t RLCNodeId, uint32 *value);  
  
int MWXDb_Get_RLCNode_nomCap(MWXCellId_t cellId,  
                              MWXObjId_t RLCNodeId, float *value);  
  
int MWXDb_Set_RLCNode_nodeId(MWXCellId_t cellId,  
                              MWXObjId_t objId, uint32 value);  
  
int MWXDb_Set_RLCNode_nomCap(MWXCellId_t cellId,  
                              MWXObjId_t objId, float value);
```

RLCPort Object

The field descriptions for the RLCPort object are listed in [Table 3-64](#).

Table 3-64 RLCPort Field Descriptions

Field	Output	Description
flag	unsigned char	Flag: first 2 bits are used to specify portNameOrObjId object's type, and 4 bits are used to represent direction of the port
filler	unsigned char	Filler
instId	int32	Instance ID
portNameOrObjId	uint32	Bridge between parasitic and netlist view
nodeObjId	int32	ID of corresponding RLC node

The prototypes for the RLCPort accessor functions are

```
int MWXDb_Get_RLCPort_flag(MWXCellId_t cellId,
                           MWXObjId_t RLCPortId, int32 *value);

int MWXDb_Get_RLCPort_filler(MWXCellId_t cellId,
                             MWXObjId_t RLCPortId, int32 *numberOfElm, int32 **filler);

int MWXDb_Get_RLCPort_instId(MWXCellId_t cellId,
                             MWXObjId_t RLCPortId, MWXObjId_t *instId);

int MWXDb_Get_RLCPort_portNameOrObjId(MWXCellId_t cellId,
                                       MWXObjId_t RLCPortId, uint32 *value);

int MWXDb_Get_RLCPort_nodeObjId(MWXCellId_t cellId,
                                MWXObjId_t RLCPortId, MWXObjId_t *nodeObjId);

int MWXDb_Set_RLCPort_flag(MWXCellId_t cellId,
                           MWXObjId_t objId, int32 value);

int MWXDb_Set_RLCPort_filler(MWXCellId_t cellId,
                             MWXObjId_t objId, int32 elm, int32 *value);

int MWXDb_Set_RLCPort_instId(MWXCellId_t cellId,
                             MWXObjId_t objId, MWXObjId_t nObjId);

int MWXDb_Set_RLCPort_portNameOrObjId(MWXCellId_t cellId,
                                       MWXObjId_t objId, uint32 value);

int MWXDb_Set_RLCPort_nodeObjId(MWXCellId_t cellId, MWXObjId_t objId,
                                MWXObjId_t nObjId);
```

RouteGuide Object

The field descriptions for the route guide database record are listed in [Table 3-65](#).

Table 3-65 RouteGuide Field Descriptions

Field	Output	Description
netId	MWXObjId_t	Database identifier of the net to be associated with the route guide
layer	int32	Layer number on which the route guide resides
dataType	int32	Number of data type on which the route guide belongs

Table 3-65 *RouteGuide Field Descriptions*

Field	Output	Description
bbox	4 of int32	Two x/y coordinate pairs representing lower-left and upper-right corners of the bounding box of the route guide

The prototypes that correspond to the field names for the route guide database record follow:

```
int MWXDb_Get_RouteGuide_netId (MWXCellId_t cellId, MWXObjId_t
    routeGuideId, MWXObjId_t *netId);

int MWXDb_Get_RouteGuide_layer (MWXCellId_t cellId, MWXObjId_t
    routeGuideId, int32 *layer);

int MWXDb_Get_RouteGuide_dataType (MWXCellId_t cellId, MWXObjId_t
    routeGuideId, int32 *dataType);

int MWXDb_Get_RouteGuide_bbox (MWXCellId_t cellId, MWXObjId_t
    routeGuideId, int32 *left, int32 *bottom, int32 *right, int32 *top);
```

Schematic Object Functions

The accessors discussed in this section provide both read and write access to the object type members defined in the MWXSch object definitions section.

SchArc Object

SchBundle Object

SchCircle Object

SchCustom Object

SchGlobalNet Object

SchJunction Object

SchLines Object

SchNameAnnotation Object

SchPinInst Object

SchPin Object

SchPolygon Object

SchPort Object

SchProperty Object

SchRectangle Object

SchRipper Object

SchSegment Object

SchSubnet Object

SchText Object

SchArc Object

The field descriptions for the SchArc object are listed in [Table 3-66](#).

Table 3-66 SchArc Object Field Descriptions

Field	Output	Description
bbox	4 of int32	Two x/y-coordinate pairs representing the lower-left and upper-right corners of the bounding box of the array
attrs	struct	Schematic drawing attributes
center	2 of int32	x/y-coordinate of circle center
width	uint32	Width of drawn arc
height	uint32	Height of drawn arc
angle1	int32	Degrees * 64 of first point on circle
angle2	int32	Degrees * 64 relative to angle1
startPoint	2 of int32	x/y-coordinate of first point on arc
middlePoint	2 of int32	x/y-coordinate of middle point on arc
endPoint	2 of int32	x/y-coordinate of end point on arc

The prototypes for the SchArc accessor functions are

```
int MWXDb_Get_SchArc_bbox (MWXCellId_t cellId, MWXObjId_t objId, int32 *
left, int32 * bottom, int32 * right, int32 * top);

int MWXDb_Get_SchArc_schAttr (MWXCellId_t cellId, MWXObjId_t objId,
MWXSchAttr_t * schAttr);

int MWXDb_Get_SchArc_center (MWXCellId_t cellId, MWXObjId_t objId,
MWXPoint_t * center);

int MWXDb_Get_SchArc_width (MWXCellId_t cellId, MWXObjId_t objId, uint32 *
value);

int MWXDb_Get_SchArc_height (MWXCellId_t cellId, MWXObjId_t objId, uint32
* value);

int MWXDb_Get_SchArc_angle1 (MWXCellId_t cellId, MWXObjId_t objId, int32 *
value);

int MWXDb_Get_SchArc_angle2 (MWXCellId_t cellId, MWXObjId_t objId, int32 *
value);

int MWXDb_Get_SchArc_startPoint (MWXCellId_t cellId, MWXObjId_t objId,
MWXPoint_t * startPoint);

int MWXDb_Get_SchArc_middlePoint (MWXCellId_t cellId, MWXObjId_t objId,
MWXPoint_t * middlePoint);

int MWXDb_Get_SchArc_endPoint (MWXCellId_t cellId, MWXObjId_t objId,
MWXPoint_t * endPoint);
```

Additional functions related to SchArcs are

- MWXDb_Create_SchArc

SchBundle Object

A SchBundle represents a bus or grouping of nets. The field descriptions for the SchBundle object are listed in [Table 3-67](#).

Table 3-67 SchBundle Object Field Descriptions

Field	Output	Description
name	string	Bundle name including dimension

The prototypes for the SchBundle accessor functions are

```
int MWXDb_Get_SchBundle_name (MWXCellId_t cellId, MWXObjId_t objId, char *
* string);

int32 MWXDb_Create_ExpBundle (const char * bundleName, MWXExpBundle_t *
expBundle);
```

```

int32 MWXDb_Free_ExpBundle (MWXExpBundle_t expBundle);
int32 MWXDb_Get_ExpBundle_width (MWXExpBundle_t expBundle, int32 * width);
int32 MWXDb_Get_ExpBundle_index (MWXExpBundle_t expBundle, const char *
name, int startIndex, int * index);

int32 MWXDb_Get_ExpBundle_element (MWXExpBundle_t expBundle, int index, char
buffer[], int bufferSize);

int32 MWXDb_Get_ExpBundle_name (MWXExpBundle_t expBundle, char name[], int
nameLength);

```

Additional functions related to SchBundles are

- MWXDb_Create_ExpBundle
- MWXDb_Free_ExpBundle
- MWXDb_Get_ExpBundle_element
- MWXDb_Get_ExpBundle_index
- MWXDb_Get_ExpBundle_name
- MWXDb_Get_ExpBundle_width

SchCircle Object

The field descriptions for the SchCircle object are listed in [Table 3-68](#).

Table 3-68 SchCircle Object Field Descriptions

Field	Output	Description
bbox	4 of int32	x/y-coordinate pairs representing bounding box
attrs	struct	Schematic drawing attributes
center	2 of int32	x/y-coordinate pair of circle center
radius	uint32	Radius

The prototypes for the SchCircle accessor functions are

```

int MWXDb_Get_SchCircle_bbox (MWXCellId_t cellId, MWXObjId_t objId, int32
* left, int32 * bottom, int32 * right, int32 * top);

int MWXDb_Get_SchCircle_schAttr (MWXCellId_t cellId, MWXObjId_t objId,
MWXSchAttr_t * schAttr);

```

```
int MWXDb_Get_SchCircle_center (MWXCellId_t cellId, MWXObjId_t objId,
MWXPoint_t * center);

int MWXDb_Get_SchCircle_radius (MWXCellId_t cellId, MWXObjId_t objId, int32
* value);
```

Additional functions related to SchCircles are

- MWXDb_Create_SchCircle

SchCustom Object

A SchCustom object represents a non-hierarchical connector such as an off-sheet connector. The field descriptions for the SchCustom object are listed in [Table 3-69](#).

Table 3-69 SchCustom Object Field Descriptions

Field	Output	Description
bbox	4 of int32	x/y-coordinate pairs representing bounding box
subNetId	objId	Subnet identifier
location	2 of int32	x/y-coordinate pair of connector location
symbolCellName	string	Cell containing symbol graphics
symbolSheet	uint8	Symbol number in symbol cell
transform	uint8	Drawing transformation code

The prototypes for the SchCustom accessor functions are

```
int MWXDb_Get_SchCustom_bbox (MWXCellId_t cellId, MWXObjId_t objId, int32
* left, int32 * bottom, int32 * right, int32 * top);

int MWXDb_Get_SchCustom_subNetId (MWXCellId_t cellId, MWXObjId_t objId,
MWXObjId_t * nObjId);

int MWXDb_Get_SchCustom_location (MWXCellId_t cellId, MWXObjId_t objId,
MWXPoint_t * location);

int MWXDb_Get_SchCustom_symbolCellName (MWXCellId_t cellId, MWXObjId_t
objId, char * * string);

int MWXDb_Get_SchCustom_symbolSheet (MWXCellId_t cellId, MWXObjId_t objId,
uint8 * symbolSheet);

int MWXDb_Get_SchCustom_transform (MWXCellId_t cellId, MWXObjId_t objId,
uint8 * transform);
```

Additional functions related to SchCustoms are

- MWXDb_Create_SchCustom

SchGlobalNet Object

The field descriptions for the SchGlobalNet object are listed in [Table 3-70](#).

Table 3-70 SchGlobalNet Object Field Descriptions

Field	Output	Description
bbox	4 of int32	x/y-coordinate pairs representing bounding box
subNetId	objId	Subnet identifier
location	2 of int32	x/y-coordinate pair of connector location
symbolCellName	string	Cell containing symbol graphics
symbolSheet	uint8	Symbol number in symbol cell
transform	uint8	Drawing transformation code

The prototypes for the SchGlobalNet accessor functions are

```
int MWXDb_Get_SchGlobalNet_bbox (MWXCellId_t cellId, MWXObjId_t objId, int32 * left, int32 * bottom, int32 * right, int32 * top);  
  
int MWXDb_Get_SchGlobalNet_subNetId (MWXCellId_t cellId, MWXObjId_t objId, MWXObjId_t * nObjId);  
  
int MWXDb_Get_SchGlobalNet_location (MWXCellId_t cellId, MWXObjId_t objId, MWXPoint_t * location);  
  
int MWXDb_Get_SchGlobalNet_symbolCellName (MWXCellId_t cellId, MWXObjId_t objId, char * * string);  
  
int MWXDb_Get_SchGlobalNet_symbolSheet (MWXCellId_t cellId, MWXObjId_t objId, uint8 * symbolSheet);  
  
int MWXDb_Get_SchGlobalNet_transform (MWXCellId_t cellId, MWXObjId_t objId, uint8 * transform);
```

Additional functions related to SchGlobalNets are

- MWXDb_Create_SchGlobalNet

SchJunction Object

The field descriptions for the SchJunction object are listed in [Table 3-71](#).

Table 3-71 SchJunction Object Field Descriptions

Field	Output	Description
bbox	4 of int32	x/y-coordinate pairs representing bounding box
subNetId	objId	Subnet identifier
location	2 of int32	x/y-coordinate pair of connector location

The prototypes for the SchJunction accessor functions are

```
int MWXDb_Get_SchJunction_bbox (MWXCellId_t cellId, MWXObjId_t objId, int32 * left, int32 * bottom, int32 * right, int32 * top);
```

```
int MWXDb_Get_SchJunction_subNetId (MWXCellId_t cellId, MWXObjId_t objId, MWXObjId_t * nObjId);
```

```
int MWXDb_Get_SchJunction_location (MWXCellId_t cellId, MWXObjId_t objId, MWXPoint_t * location);
```

Additional functions related to SchJunctions are

- MWXDb_Create_SchJunction

SchLines Object

The field descriptions for the SchLines object are listed in [Table 3-72](#).

Table 3-72 SchLines Object Field Descriptions

Field	Output	Description
bbox	4 of int32	x/y-coordinate pairs representing bounding box
attrs	struct	Schematic drawing attributes
numberOfPoint	uint32	Number of points in points array
points	2n of int32	x/y-coordinate pairs of each point in polyline

The prototypes for the SchLines accessor functions are

```
int MWXDb_Get_SchLines_bbox (MWXCellId_t cellId, MWXObjId_t objId, int32 *
left, int32 * bottom, int32 * right, int32 * top);

int MWXDb_Get_SchLines_schAttr (MWXCellId_t cellId, MWXObjId_t objId,
MWXSchAttr_t * schAttr);

int MWXDb_Get_SchLines_points (MWXCellId_t cellId, MWXObjId_t objId, int32
* numOfPoint, MWXPoint_t * * points);
```

Additional functions related to SchLines are

- MWXDb_Create_SchLines

SchNameAnnotation Object

The field descriptions for the SchNameAnnotation object are listed in [Table 3-73](#).

Table 3-73 SchNameAnnotation Object Field Descriptions

Field	Output	Description
bbox	4 of int32	x/y-coordinate pairs representing bounding box
location	2 of int32	x/y-coordinate pair
textHeight	uint32	Text height
refObjId	objId	Object whose name is displayed
status	uint8	Visible, etc.

The prototypes for the SchNameAnnotation accessor functions are

```
int MWXDb_Get_SchNameAnnotation_schAttr (MWXCellId_t cellId, MWXObjId_t
objId, MWXSchAttr_t * schAttr);

int MWXDb_Get_SchNameAnnotation_bbox (MWXCellId_t cellId, MWXObjId_t objId,
int32 * left, int32 * bottom, int32 * right, int32 * top);

int MWXDb_Get_SchNameAnnotation_location (MWXCellId_t cellId, MWXObjId_t
objId, MWXPoint_t * location);

int MWXDb_Get_SchNameAnnotation_textHeight (MWXCellId_t cellId, MWXObjId_t
objId, int32 * value);

int MWXDb_Get_SchNameAnnotation_refObjId (MWXCellId_t cellId, MWXObjId_t
objId, MWXObjId_t * nObjId);
```

Additional functions related to SchNameAnnotations are

- MWXDb_Create_SchNameAnnotation

SchPinInst Object

A SchPinInst object is a 0x implementation of a pin in the child cell placed by the attached instance. The field descriptions for the SchPinInst object are listed in [Table 3-74](#).

Table 3-74 SchPinInst Object Field Descriptions

Field	Output	Description
bbox	4 of int32	x/y-coordinate pairs representing bounding box
subNetId	objId	Subnet identifier
location	2 of int32	x/y-coordinate pair of connector location
transform	uint8	Drawing transformation code
cellInstId	objId	Cell instance this object is attached to
direction	uint8	Connection direction

The prototypes for the SchPinInst accessor functions are

```
int MWXDb_Get_SchPinInst_bbox (MWXCellId_t cellId, MWXObjId_t objId, int32 * left, int32 * bottom, int32 * right, int32 * top);  
  
int MWXDb_Get_SchPinInst_subNetId (MWXCellId_t cellId, MWXObjId_t objId, MWXObjId_t * nObjId);  
  
int MWXDb_Get_SchPinInst_location (MWXCellId_t cellId, MWXObjId_t objId, MWXPoint_t * location);  
  
int MWXDb_Get_SchPinInst_transform (MWXCellId_t cellId, MWXObjId_t objId, int32 * value);  
  
int MWXDb_Get_SchPinInst_cellInstId (MWXCellId_t cellId, MWXObjId_t objId, MWXObjId_t * nObjId);  
  
int MWXDb_Get_SchPinInst_direction (MWXCellId_t cellId, MWXObjId_t objId, MWXPortDirection_e *direction);
```

Additional functions related to SchPinInsts are

- MWXDb_SetPinInstInParallel

- MWXDb_Get_PinInstInstanceDimension
- MWXDb_Get_PinNetOrBundleByPinInst

SchPin Object

The field descriptions for the SchPin object are listed in [Table 3-75](#).

Table 3-75 SchPin Object Field Descriptions

Field	Output	Description
bbox	4 of int32	x/y-coordinate pairs representing bounding box
netOrBundleId	objId	Net or Bundle objId
location	2 of int32	x/y-coordinate pair of connector location
transform	uint8	Drawing transformation code
direction	uint8	Connection direction

The prototypes for the SchPin accessor functions are

```
int MWXDb_Get_SchPin_bbox (MWXCellId_t cellId, MWXObjId_t objId, int32 *
left, int32 * bottom, int32 * right, int32 * top);

int MWXDb_Get_SchPin_netOrBundleId (MWXCellId_t cellId, MWXObjId_t objId,
MWXObjId_t * nObjId);

int MWXDb_Get_SchPin_location (MWXCellId_t cellId, MWXObjId_t objId,
MWXPoint_t * location);

int MWXDb_Get_SchPin_transform (MWXCellId_t cellId, MWXObjId_t objId, int32
* value);

int MWXDb_Get_SchPin_direction (MWXCellId_t cellId, MWXObjId_t objId,
MWXPortDirection_e *direction);
```

Additional functions related to SchPins are

- MWXDb_Create_SchPin
- MWXDb_Get_First_SchPin

SchPolygon Object

The field descriptions for the SchPolygon object are listed in [Table 3-76](#).

Table 3-76 SchPolygon Object Field Descriptions

Field	Output	Description
bbox	4 of int32	x/y-coordinate pairs representing bounding box
attr	struct	Schematic drawing attributes
points	2n of int32	Connected points in the polygon

The prototypes for the SchPolygon accessor functions are

```
int MWXDb_Get_SchPolygon_bbox (MWXCellId_t cellId, MWXObjId_t objId, int32 * left, int32 * bottom, int32 * right, int32 * top);
```

```
int MWXDb_Get_SchPolygon_schAttr (MWXCellId_t cellId, MWXObjId_t objId, MWXSchAttr_t * schAttr);
```

```
int MWXDb_Get_SchPolygon_points (MWXCellId_t cellId, MWXObjId_t objId, int32 * numOfPoint, MWXPoint_t * * points);
```

Additional functions related to SchPolygons are

- MWXDb_Create_SchPolygon

SchPort Object

The field descriptions for the SchPort object are listed in [Table 3-77](#).

Table 3-77 SchPort Object Field Descriptions

Field	Output	Description
bbox	4 of int32	x/y-coordinate pairs representing bounding box
subNetId	objId	Subnet identifier
location	2 of int32	x/y-coordinate pair of connector location
symbolCellName	string	Cell containing symbol graphics

Table 3-77 SchPort Object Field Descriptions

Field	Output	Description
symbolSheet	uint8	Symbol number in symbol cell
transform	uint8	Drawing transformation code
direction	uint8	Connection direction

The prototypes for the SchPort accessor functions are

```
int MWXDb_Get_SchPort_bbox (MWXCellId_t cellId, MWXObjId_t objId, int32 *
left, int32 * bottom, int32 * right, int32 * top);

int MWXDb_Get_SchPort_subNetId (MWXCellId_t cellId, MWXObjId_t objId,
MWXObjId_t * nObjId);

int MWXDb_Get_SchPort_location (MWXCellId_t cellId, MWXObjId_t objId,
MWXPoint_t * location);

int MWXDb_Get_SchPort_symbolCellName (MWXCellId_t cellId, MWXObjId_t objId,
char * * string);

int MWXDb_Get_SchPort_symbolSheet (MWXCellId_t cellId, MWXObjId_t objId,
uint8 * symbolSheet);

int MWXDb_Get_SchPort_transform (MWXCellId_t cellId, MWXObjId_t objId, uint8
* transform);

int MWXDb_Get_SchPort_direction (MWXCellId_t cellId, MWXObjId_t objId,
MWXPortDirection_e *direction);
```

Additional functions related to SchPorts are

- MWXDb_Create_SchPortOnSheet
- MWXDb_Create_SchPortOnSubnet

SchProperty Object

The field descriptions for the SchProperty object are listed in [Table 3-78](#).

Table 3-78 SchProperty Object Field Descriptions

Field	Output	Description
location	2 of int32	x/y-coordinate pair of location
locked	boolean	Locked state

Table 3-78 SchProperty Object Field Descriptions

Field	Output	Description
textHeight	uint8	Text height
ownerId	objId	Property owner (instance, port, 0, etc)
propName	string	Property name

The prototypes for the SchProperty accessor functions are

```
int32 MWXDb_Get_SchProperty_locked (MWXCellId_t cellId, MWXObjId_t objId,
MWXBoolean_e * locked);
```

```
int32 MWXDb_Get_SchProperty_location (MWXCellId_t cellId, MWXObjId_t objId,
MWXPoint_t * location);
```

```
int32 MWXDb_Get_SchProperty_textHeight (MWXCellId_t cellId, MWXObjId_t
objId, int32 * value);
```

```
int32 MWXDb_Get_SchProperty_ownerId (MWXCellId_t cellId, MWXObjId_t objId,
MWXObjId_t * nObjId);
```

```
int32 MWXDb_Get_SchProperty_propName (MWXCellId_t cellId, MWXObjId_t objId,
char * * string);
```

Additional functions related to SchProperty* are

- MWXDb_Create_SchProperty
- MWXDb_Get_SchProperty_ById
- MWXDb_Get_SchProperty_ByName

SchRectangle Object

The field descriptions for the SchRectangle object are listed in [Table 3-79](#).

Table 3-79 SchRectangle Object Field Descriptions

Field	Output	Description
bbox	4 of int32	x/y-coordinate pairs representing bounding box
attrs	struct	Schematic drawing attributes

The prototypes for the SchRectangle accessor functions are

```
int MWXDb_Get_SchRectangle_bbox (MWXCellId_t cellId, MWXObjId_t objId, int32
* left, int32 * bottom, int32 * right, int32 * top);

int MWXDb_Get_SchRectangle_schAttr (MWXCellId_t cellId, MWXObjId_t objId,
MWXSchAttr_t * schAttr);
```

Additional functions related to SchRectangles are

- MWXDb_Create_SchRectangle

SchRipper Object

The field descriptions for the SchRipper object are listed in [Table 3-80](#).

Table 3-80 SchRipper Object Field Descriptions

Field	Output	Description
connectorId1	objId	First connector
connectorId2	objId	Second connector

The prototypes for the SchRipper accessor functions are

```
int MWXDb_Get_SchRipper_connectorId1 (MWXCellId_t cellId, MWXObjId_t objId,
MWXObjId_t * nObjId);

int MWXDb_Get_SchRipper_connectorId2 (MWXCellId_t cellId, MWXObjId_t objId,
MWXObjId_t * nObjId);
```

Additional functions related to SchRippers are

- MWXDb_Create_SchRipper

SchSegment Object

The field descriptions for the SchSegment object are listed in [Table 3-81](#).

Table 3-81 SchSegment Object Field Descriptions

Field	Output	Description
connectorId1	objId	First connector
connectorId2	objId	Second connector

The prototypes for the SchSegment accessor functions are


```
int MWXDb_Get_SchSegment_connectorId1 (MWXCellId_t cellId, MWXObjId_t
objId, MWXObjId_t * nObjId);

int MWXDb_Get_SchSegment_connectorId2 (MWXCellId_t cellId, MWXObjId_t
objId, MWXObjId_t * nObjId);
```

Additional functions related to SchSegments are

- MWXDb_Create_SchSegment

SchSubnet Object

The field descriptions for the SchSubnet object are listed in [Table 3-82](#).

Table 3-82 SchSubnet Object Field Descriptions

Field	Output	Description
bbox	4 of int32	x/y-coordinate pairs representing bounding box
netId	objId	Net identifier
attrs	struct	Schematic drawing attributes

The prototypes for the SchSubnet accessor functions are

```
int MWXDb_Get_SchSubnet_bbox (MWXCellId_t cellId, MWXObjId_t objId, int32
* left, int32 * bottom, int32 * right, int32 * top);

int MWXDb_Get_SchSubnet_netId (MWXCellId_t cellId, MWXObjId_t objId,
MWXObjId_t * nObjId);

int MWXDb_Get_SchSubnet_schAttr (MWXCellId_t cellId, MWXObjId_t objId,
MWXSchAttr_t * schAttr);
```

Additional functions related to SchSubnets are

- MWXDb_AddSubnetToNetOrBundle
- MWXDb_Create_SchSubnet
- MWXDb_GetOrCreate_SchNet

SchText Object

The field descriptions for the SchText object are listed in [Table 3-83](#).

Table 3-83 SchText Object Field Descriptions

Field	Output	Description
bbox	4 of int32	x/y-coordinate pairs representing bounding box
attrs	struct	Schematic drawing attributes
text	string	Text string
textHeight	uint8	Text height
location	2 of int32	Location

The prototypes for the SchText accessor functions are

```
int MWXDb_Get_SchText_bbox (MWXCellId_t cellId, MWXObjId_t objId, int32 *  
left, int32 * bottom, int32 * right, int32 * top);
```

```
int MWXDb_Get_SchText_schAttr (MWXCellId_t cellId, MWXObjId_t objId,  
MWXSchAttr_t * schAttr);
```

```
int MWXDb_Get_SchText_text (MWXCellId_t cellId, MWXObjId_t objId, char * *  
string);
```

```
int MWXDb_Get_SchText_textHeight (MWXCellId_t cellId, MWXObjId_t objId,  
int32 * value);
```

```
int MWXDb_Get_SchText_location (MWXCellId_t cellId, MWXObjId_t objId,  
MWXPoint_t * location);
```

Additional functions related to SchTexts are

- MWXDb_Create_SchText

Text Object

The field descriptions for the Text object are listed in [Table 3-84](#).

Table 3-84 Text Object Field Descriptions

Field	Output	Description
bbox	4 of int32	Two x/y-coordinate pairs representing the lower-left and upper-right corners of the bounding box of the text.
string	char*	Text string.
position	MWXPoint_t*	Specification of the x/y-coordinate pair defining location.
text_height	uint32	Height of the text, in database units.
font_id	int32	A number from 0 to 255 indicating the font used for the text.
tjust	int32	Bit-field indicating the text justification relative to the position; bit-and with one of the following constants to check the value: dbcLowerLeft dbcLowerCenter dbcLowerRight dbcCenterLeft dbcCenterCenter dbcCenterRight dbcUpperLeft dbcUpperCenter dbcUpperRight
layer	int32	Number of the layer on which the text resides (layer numbers appear in the technology file).
transformCode	int32	See “transformCode” on page 3-111 .

The prototypes for the Text accessor functions are

```
int MWXDb_Get_Text_bbox (MWXCellId_t cellId,  
    MWXObjId_t objId, int32 *left, int32 *bottom, int32 *right, int32 *top);  
  
int MWXDb_Get_Text_string (MWXCellId_t cellId,  
    MWXObjId_t objId, char **string);
```

```

int MWXDb_Get_Text_position (MWXCellId_t cellId,
                             MWXObjId_t objId, MWXPoint_t **point);

int MWXDb_Get_Text_text_height (MWXCellId_t cellId,
                                MWXObjId_t objId, uint32 *value);

int MWXDb_Get_Text_font_id (MWXCellId_t cellId,
                             MWXObjId_t objId, int32 *value);

int MWXDb_Get_Text_tjust (MWXCellId_t cellId,
                           MWXObjId_t objId, int32 *value);

int MWXDb_Get_Text_layer (MWXCellId_t cellId,
                           MWXObjId_t objId, int32 *value);

int MWXDb_Get_Text_transformCode (MWXCellId_t cellId,
                                  MWXObjId_t objId, int32 *value);

```

Additional functions related to Text are

- [MWXDb_Create_Text](#)
- [MWXDb_Replace_CellTextChar](#)
- [MWXDb_Replace_CellTextString](#)
- [MWXDb_Truncate_CellText](#)
- [MWXDb_Truncate_Text](#)

TilePattern Object

The field descriptions for the TilePattern object are listed in [Table 3-85](#).

Table 3-85 TilePattern Object Field Descriptions

Field	Output	Description
dataValues	int32	Array of data values
name	char*	Name of the tilePattern
numberOfTile	int32	Number of tile in the tilePattern object
numberOfDataValue	int32	Number of data value in the tilePattern object
tilePatternNumber	int32	The tile pattern number used to get the pattern name

The prototypes for the TilePattern accessor functions are

```
int MWXDb_Get_TilePattern_dataValues (MWXCellId_t cellId,  
    MWXObjId_t objId, int32 *numOfElm, int32 **value);  
  
int MWXDb_Get_TilePattern_name (MWXCellId_t cellId,  
    MWXObjId_t objId, char**name);  
  
int MWXDb_Get_TilePattern_numberOfTile (MWXCellId_t cellId,  
    MWXObjId_t objId, int32 *name);  
  
int MWXDb_Get_TilePattern_numberOfDataValue (MWXCellId_t cellId,  
    MWXObjId_t objId, int32 *name);  
  
int MWXDb_Get_TilePattern_tilePatternNumber (MWXCellId_t cellId,  
    MWXObjId_t objId, int32 *name);
```

See also

- [MWXDb_Create_TilePattern](#)

VerticalWire Object

The field descriptions for the VerticalWire object are listed in [Table 3-86](#).

Table 3-86 VerticalWire Object Field Descriptions

Field	Output	Description
netId	MWXObjId_t	Database identifier of the net associated with the wire.
wireMasterIdx	int32	Index value of associated WireMaster object (see index field of “WireMaster Object” on page 3-102).
layer	int32	Number of the layer on which the wire resides. (Layer numbers appear in the technology file.)
regionNumber	int32	Zero if the wire is not associated with a region; nonzero and the regionNumber object represent the identifier of the region with which the wire is associated.
startPoint	MWXPoint_t*	Specification of the x/y-coordinates of the starting point of the wire.

Table 3-86 VerticalWire Object Field Descriptions

Field	Output	Description
length	int32	Length, in database units, of the wire; a negative number means that the wire extends down from the start point while a positive number means the wire extends up.
routeType	int32	See “routeType” on page 3-108 .

The prototypes for the VerticalWire accessor functions are

```
int MWXDb_Get_VerticalWire_netId (MWXCellId_t cellId,
    MWXObjId_t objId, MWXObjId_t *nObjId);

int MWXDb_Get_VerticalWire_wireMasterIdx (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_VerticalWire_layer (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_VerticalWire_regionNumber (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_VerticalWire_startPoint (MWXCellId_t cellId,
    MWXObjId_t objId, MWXPoint_t **point);

int MWXDb_Get_VerticalWire_length (MWXCellId_t cellId,
    MWXObjId_t objId, int32 *value);

int MWXDb_Get_VerticalWire_routeType(MWXCellId_t CellId,
    MWXObjId_t objId, int32 routeType);
```

See also

- [MWXDb_Set_VerticalWire_routeType](#)

ViaRegion Object

The field descriptions for the ViaRegion object are listed in [Table 3-87](#).

Table 3-87 ViaRegion Object Field Descriptions

Field	Output	Description
number of point	int32	Number of points used to define the ViaRegion object

Table 3-87 ViaRegion Object Field Descriptions (Continued)

Field	Output	Description
Points	MWXPoint_t Array	An array of the x/y-coordinate pairs of points
portId	MWXObjId_t	Database identifier of the related port
ContactCodeIdx	int32*	The contact code index
bbox	4 of int32	Two x/y-coordinate pairs representing the lower-left and upper-right corners of the bounding box of the rectangle
dataType	int32	GDSII data type
rotate90	int32	Rotate flag
userMarked	int32	User-marked flag

The prototypes for the ViaRegion accessor functions are

```
int MWXDb_Get_ViaRegion_numberOfPoint ((MWXCellId_t cellId,
                                         MWXObjId_t objId, int32 *value));

int MWXDb_Get_ViaRegion_points ((MWXCellId_t cellId,
                                  MWXObjId_t objId, int32 *numOfPoint, MWXPoint_t **points));

int MWXDb_Get_ViaRegion_portId ((MWXCellId_t cellId,
                                  MWXObjId_t objId, MWXObjId_t *nObjId));

int MWXDb_Get_ViaRegion_contactCodeIdx ((MWXCellId_t cellId,
                                          MWXObjId_t objId, int32 *value));

int MWXDb_Get_ViaRegion_bbox ((MWXCellId_t cellId,
                               MWXObjId_t objId, int32 *left, int32 *bottom, int32 *right, int32 *top));

int MWXDb_Get_ViaRegion_dataType (MWXCellId_t cellId,
                                   MWXObjId_t objId, int32 *dataType);

int MWXDb_Get_ViaRegion_rotate90 (MWXCellId_t cellId,
                                   MWXObjId_t objId, int32 *rotate90);

int MWXDb_Get_ViaRegion_userMarked (MWXCellId_t cellId,
                                     MWXObjId_t objId, int32 *userMarked);
```

See also

- [MWXDb_Get_ViaRegionFromPort](#)

WireDirection Object

The field descriptions for the WireDirection object are listed in [Table 3-88](#).

Table 3-88 WireDirection Object Field Descriptions

Field	Output	Description
preferredDirection	int32	0 for horizontal, 1 for vertical, 2 if not specified.
layer	int32	Number of the layer on which wires routed in the preferred direction will reside. (Layer numbers appear in the technology file.)

The prototypes for the WireDirection accessor functions are

```
int MWXDb_Get_WireDirection_preferedDirection (MWXCellId_t cellId,  
                                                MWXObjId_t objId, int32 *value);
```

```
int MWXDb_Get_WireDirection_layer (MWXCellId_t cellId,  
                                    MWXObjId_t objId, int32 *value);
```

For more information about WireDirection accessor functions, see
[“MWXDb_Get_WireDirection_preferedDirection” on page 4-223](#) and
[“MWXDb_Get_WireDirection_layer” on page 4-222](#).

See also

- [MWXDb_Create_WireDirection](#)

WireMaster Object

The field descriptions for the WireMaster object are listed in
[Table 3-89](#).

Table 3-89 WireMaster Object Field Descriptions

Field	Output	Description
index	int32	Unique identifier for the WireMaster object.
layer	int32	Number of the layer on which the wires associated with the wire master reside. (Layer numbers appear in the technology file.)

Table 3-89 WireMaster Object Field Descriptions

Field	Output	Description
pathType	int32	0 (bit 0 is set) if the path edge aligns with the coordinate; 2 (bit 1 is set) if the path edge extends a half-width beyond the coordinate; bit-and with the constant dbcSquareEnds or dbcSquareEndsByHalfWidth to check the value.
width	uint32	Width, in database units, of wires associated with the WireMaster object.

The prototypes for the WireMaster accessor functions are

```
int MWXDb_Get_WireMaster_index (MWXCellId_t cellId,
                                MWXObjId_t objId, int32 *value);

int MWXDb_Get_WireMaster_layer (MWXCellId_t cellId,
                                MWXObjId_t objId, int32 *value);

int MWXDb_Get_WireMaster_pathType (MWXCellId_t cellId,
                                    MWXObjId_t objId, int32 *value);

int MWXDb_Get_WireMaster_width (MWXCellId_t cellId,
                                MWXObjId_t objId, int32 *value);
```

Additional functions related to WireMaster are

- MWXDb_Get_GLink_wireMasterIdx (see [Glink Object](#))
- MWXDb_Get_HorizontalWire_wireMasterIdx (see [HorizontalWire Object](#))
- MWXDb_Get_VerticalWire_wireMasterIdx (see [VerticalWire Object](#))
- [MWXDb_Create_Horizontal_Wire](#)
- [MWXDb_Create_Vertical_Wire](#)
- [MWXDb_Get_WireMasterId](#)

WireTrack Object

The field descriptions for the WireTrack object are listed in [Table 3-90](#).

Table 3-90 WireTrack Object Field Descriptions

Field	Output	Description
trackDirection	int32	0 if the track is vertical, 1 otherwise.
layer	int32	Number of the layer on which the wire track resides. (Layer numbers appear in the technology file.)
type	int32	0 if the track is a wire track, 1 if it's a via track.
bbox	4 of int32	Two x/y-coordinate pairs representing the lower-left and upper-right corners of the bounding box of the wire track.
startTrack	int32	Coordinate of the first track (y-coordinate for horizontal tracks, x-coordinate for vertical tracks).
trackStep	int32	Distance between tracks.
times	int32	Number of tracks.

The prototypes for the WireTrack accessor functions are

```
int MWXDb_Get_WireTrack_trackDirection (MWXCellId_t cellId,  
                                         MWXObjId_t objId, int32 *value);  
  
int MWXDb_Get_WireTrack_layer (MWXCellId_t cellId,  
                               MWXObjId_t objId, int32 *value);  
  
int MWXDb_Get_WireTrack_type (MWXCellId_t cellId,  
                              MWXObjId_t objId, int32 *value);  
  
int MWXDb_Get_WireTrack_bbox (MWXCellId_t cellId,  
                              MWXObjId_t objId, int32 *left, int32 *bottom, int32 *right, int32 *top);  
  
int MWXDb_Get_WireTrack_startTrack (MWXCellId_t cellId,  
                                     MWXObjId_t objId, int32 *value);  
  
int MWXDb_Get_WireTrack_trackStep (MWXCellId_t cellId,  
                                    MWXObjId_t objId, int32 *value);  
  
int MWXDb_Get_WireTrack_times (MWXCellId_t cellId,  
                               MWXObjId_t objId, int32 *value);
```

For more information, see

- [MWXDb_Get_WireTrack_trackDirection](#)
- [MWXDb_Get_WireTrack_layer](#)
- [MWXDb_Get_WireTrack_type](#)
- [MWXDb_Get_WireTrack_bbox](#)
- [MWXDb_Get_WireTrack_startTrack](#)
- [MWXDb_Get_WireTrack_trackStep](#)
- [MWXDb_Get_WireTrack_times](#)

Common Object Fields

This section lists the common object fields, which are referenced in the object field tables.

direction

Direction is defined by an 8-bit field. Bits 0 through 3 specify port direction. Constant values are supplied for setting or testing the value. [Table 3-91](#) describes the direction field.

Table 3-91 direction Field

Value	Constant Mask	Description
0	MWXInputPort	Bit 0 is set to 1 for an input port.
1	MWXOutputPort	Bit 1 is set to 1 for an output port.
2	MWXInputOutputPort	Bit 2 is set to 1 for an input/output port.
3	MWXTristatePort	Bit 3 is set to 1 for a three-state port.

Additional information about ports is described in [Table 3-92](#).

Table 3-92 port field

Value	Constant Mask	Description
1	MWXPortShouldTieHigh	The port is tied high.

Table 3-92 *port field*

Value	Constant Mask	Description
2	MWXPortShouldTieLow	The port is tied low.
3	MWXIsVDDPort	The port is for power connections.
4	MWXIsGNDPort	The port is for ground connections.

accessDirection

Access direction is defined by an 8-bit field. Bits 0 through 3 specify the access direction. Constant values are supplied for setting or testing the value. [Table 3-93](#) describes the accessDirection field.

Table 3-93 *accessDirection Field*

Value	Description
0	Bit 0 is set to 1 to indicate right access.
1	Bit 1 is set to 1 to indicate left access.
2	Bit 2 is set to 1 to indicate top access.
3	Bit 3 is set to 1 to indicate bottom access.

cellFlag

The cellFlag is a bit field that indicates the cell type. [Table 3-94](#) describes the meaning of the individual bits and identifies the name of the constant mask to use to check a bit value. To check the value, bit-and the constant with the cellFlag value.

Table 3-94 *cellFlag Bit Fields*

Value	Constant Mask	Description
3	MWXcMaskLayoutViewType	The cell contains graphic mask data. It might or might not have netlist information.

Table 3-94 *cellFlag Bit Fields (Continued)*

Value	Constant Mask	Description
4	MWXcNetListViewType	The cell is a netlist-only cell, generally the result of an edit of Verilog translation. No graphics will be present.
5	MWXcSchematicViewType	Not used.
6	MWXcSymbolicViewType	Not used.
8	MWXcArcPlanCell	The cell is in the floorplanning phase.
9	MWXcLogicModelViewType	The cell contains timing information and usually has a view name of *.TIM.
10	MWXcArcChipSubType	The cell was generated by Astro and is a fixed-die standard cell design.
13	MWXcIOPadCellSubType	The cell is a pad.
14	MWXcStandardCellSubType	The cell is a standard cell. <i>Module cell</i> is the equivalent term for a gate array design.
15	MWXcCornerPadCellSubType	The cell is a corner pad.
16	MWXcMacroCellSubType	The cell is a hierarchical block.
17	MWXcChipCellSubType	The cell is a top-level design with bonding pads.
18	MWXcImageCellSubType	Identifies an underlying base array cell
30	MWXcSpecialViaCellSubType	The cell is used as a special via in gate arrays.
32	MWXcFlipChipPadCellSubType	Used in conjunction with dbbIOPadCell (bit 13) to identify a Flip-Chip pad.

recType

The recType field specifies the record types used in the PortToPort table and the PortToPortDelay table. [Table 3-95](#) describes the recType bit fields.

Table 3-95 recType Bit Fields

Value	Constant Mask	Description
0	MWXcDefaultDelayType	Default delay type
1	MWXcStartCondDelayType	Start condition delay type
2	MWXcEndCondDelayType	End condition delay type
3	MWXcStartEndCondDelayType	Start and end condition delay type
4	MWXcEnaCondDelayType	Enable condition delay type
8	MWXc3dimDelayType	Three-dimensional delay type
9	MWXc3dimStartCondDelayType	Three-dimensional start condition delay type
10	MWXc3dimEndCondDelayType	Three-dimensional end condition delay type
11	MWXc3dimStartEndCondDelayType	Three-dimensional start and end condition delay type

routeType

Specifies the types of operations created by the object. Only bits 1 through 5 are used in the routeType definition.

Note: If you have bit 0 turned on, you need to mask bits 1-5 and ignore bit 0 to get a correct routeType value. For example:

routeType & 0x3E (hex value same as 111110 in binary).

[Table 3-96](#) describes the routeType bit fields.

Table 3-96 routeType Bit Fields

Value	Constant Mask	Description
0	MWXcSignalUser	Default type for shapes created by editor.
2	MWXcSignalGlobal	Used for communication between the global router and detail router
4	MWXcSignalDetail	Generated by the detail router
16	MWXcPGRing	Generated by the prerouter
18	MWXcPGStrap	Generated by the prerouter
20	MWXcPGMacroIOConn	Connection from macro/Pad to Ring/strap
22	MWXcPGStdcellConn	PG rail over standard cell
32	MWXcCLKRing	Generated by the prerouter clock route type
34	MWXcCLKStrap	Generated by the prerouter clock route type
36	MWXcCLKZeroSkew	Generated by the zero-skew router
48	MWXcMiscBus	Generated by the bus router

statusFlag

The statusFlag is a bit field that provides additional pin information. [Table 3-97](#) describes the statusFlag bit fields.

Table 3-97 statusFlag Bit Fields

Value	Description
0	Set if the pin is placed
1	Set if the pin is fixed
2	Set if the pin is a feedthrough
3	Set if the pin is straight (for pins)

Table 3-97 *statusFlag Bit Fields (Continued)*

Value	Description
3	Set if pin is a bus-region pin (for region pins)
4	Set if the pin is defined by the user
5	Set if the pin is a power/ground rail pin
6	Set if the pin is a strip pin
7	Set if the pin is a ring pin

tableType

The tableType field specifies the types of timing tables. [Table 3-98](#) describes the tableType bit fields.

Table 3-98 *tableType Bit Fields*

Value	Constant	Description
1	MWXcRisePropagationTable	Defines the propagation delay of an arc when the output signal is rising
2	MWXcFallPropagationTable	Defines the propagation delay of an arc when the output signal is falling
3	MWXcRiseTransitionTable	Defines the transition delay of an arc when the output signal is rising
4	MWXcFallTransitionTable	Defines the transition delay of an arc when the output signal is falling
5	MWXcRiseConstraintTable	Defines the delay constraint of an arc when the output signal is rising
6	MWXcFallConstraintTable	Defines the delay constraint of an arc when the output signal is falling
7	MWXcRiseCellDelayTable	Defines the cell delay of an arc when the output signal is rising

Table 3-98 *tableType Bit Fields (Continued)*

Value	Constant	Description
8	MWXcFallCellDelayTable	Defines the cell delay of an arc when the output signal is falling
9	MWXcSwitchingPowerTable	Defines the power consumed by the outputs switching in response to input switching
10	MWXcInternalPowerTable	Defines the power consumed when the cell inputs switch but the outputs do not
11	MWXcLeakagePowerTable	Defines the power dissipated by leakage currents

transformCode

The transformation code specifies the orientation of the cell instance (rotated, flipped, and so on). Constant values are supplied for setting or testing the value. Note that there are multiple sets of transformations that may be used to obtain the same orientation (such as MWXcRotate0MirrorX and MWXcRotate180MirrorY in [Table 3-99](#)).

Table 3-99 *transformCode Field*

Value	Constant	Description	Orientation
0	MWXcRotate0	No transformation	
2	MWXcRotate90	Rotated 90 degrees counterclockwise	
4	MWXcRotate180	Rotated 180 degrees counterclockwise	
6	MWXcRotate270	Rotated 270 degrees counterclockwise	
12	MWXcRotate0MirrorX	Flipped on the y-axis	

Table 3-99 *transformCode* Field (Continued)

Value	Constant	Description	Orientation
10	MWXcRotate90MirrorX	Flipped on the x-axis and rotated 90 degrees counterclockwise	└┐
8	MWXcRotate180MirrorX	Flipped on the x-axis	└┐
14	MWXcRotate270MirrorX	Flipped on the y-axis and rotated 90 degrees counterclockwise	┐└
8	MWXcRotate0MirrorY	Flipped on the x-axis	└┐
14	MWXcRotate90MirrorY	Flipped on the y-axis and rotated 90 degrees counterclockwise	┐└
12	MWXcRotate180MirrorY	Flipped on the y-axis	┐└
10	MWXcRotate270MirrorY	Flipped on the x-axis and rotated 90 degrees counterclockwise	└┐

4

Database Functions

This chapter describes the Milkyway API (`MWxdb_*`) database functions. Use these functions to create processes that perform operations similar to those the Scheme commands perform.

MWXdB_Abort_Cell

Closes a cell without saving your changes.

Syntax

```
int MWXdB_Abort_Cell (
    MWXCellId_t cellId /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell.

Return

MWXNoLibraryOpened	Occurs if the library is not open.
MWXSuccess	Occurs if the cell is closed successfully.
MWXFail	Occurs if the cell is not closed successfully.

Example

```
int AbortCell (MWXCellId_t cellId) {
    int rc ;
    MWX_Printf("INFO -- MWXdB_Abort_Cell\n");

    rc = MWXdB_Abort_Cell(cellId);
    switch (rc) {
case MWXSuccess:
    MWX_Printf("Successfully Abort the cell\n");
    return (MWXSuccess);
case MWXFail:
    MWX_Printf("ERROR -- Failed to operate function MWXdB_Abort_Cell\n");
    return (MWXFail);
case MWXNoLicense:
    MWX_Printf("ERROR -- Failed No License\n");
    return (MWXFail);
    }
}
```

Scheme Equivalent

dbAbortCell

(see Physical Implementation Online Help)

MWXDb_Add_CellInstance_ToCellRegion

Adds the cell instance identified by `cellInstanceId` to the region identified by `cellRegionId`.

Syntax

```
MWXDb_Add_CellInstance_ToCellRegion(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t cellRegionId, /* INPUT */  
    MWXObjId_t cellInstanceId /* INPUT */  
)
```

Parameters

<code>cellId</code>	Specifies the database ID of the cell containing the region and the cell instance.
<code>cellRegionId</code>	Specifies the database ID of the region in which to add the instance.
<code>cellInstanceId</code>	Specifies the database ID of the instance to add to the region.

Return

<code>MWXSuccess</code>	Occurs if the cell instance is successfully added to the region.
<code>MWXFail</code>	Occurs if the cell instance is not successfully added to the region.

Example

```
int mwAddCellInstanceToCellRegion (MWXCellId_t cellId,  
                                   MWXObjId_t cellRegionId,  
                                   MWXObjId_t cellInstanceId)  
{  
    int rc;  
  
    rc = MWXDb_Add_CellInstance_ToCellRegion (cellId, cellRegionId,  
cellInstanceId);  
    switch (rc) {  
        case MWXSuccess:  
            MWX_Printf("Successfully add CellInstance %d to  
CellRegion %d\n", cellInstanceId, cellRegionId);  
            return (MWXSuccess);  
        case MWXFail:  
            MWX_Printf("ERROR -- Failed to add instance to region\n");  
            return (MWXFail);  
        case MWXNoLicense:  
            MWX_Printf("ERROR -- Failed No License\n");  
            return (MWXFail);  
        default:  
            MWX_Printf("ERROR -- unknown return code\n");  
            return (MWXFail);  
    }  
}
```

```
}
}
```

Scheme Equivalent

dbAddCellInstToCellRegion
(see Physical Implementation Online Help)

MWXDb_Add_CellInst_ToPlanGroup_ByName

Adds the named cell instance to the Jupiter group.

Syntax

```
int MWXDb_Add_CellInst_ToPlanGroup_ByName (
    MWXCellId_t cellId, /* INPUT */
    char *cellInstName, /* INPUT */
    MWXObjId_t groupId /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that owns group and cellInst.
cellInstName	Specifies a cellInst name.
groupId	Specifies the database identifier of the group.

Return

MWXNoLicense	Occurs if no software license is available.
MWXSuccess	Occurs if the cell instance is successfully added to the group.
MWXFail	Occurs if the cell instance cannot be successfully added to the group.

Example

```
int mwAddCellInstToPlanGroupByName (MWXCellId_t cellId,
char *cellInstName,
MWXObjId_t groupId)
{
    int rc;

    MWX_Printf("INFO -- test MWXDb_Add_CellInst_ToPlanGroup_ByName ...\n");

    rc = MWXDb_Add_CellInst_ToPlanGroup_ByName(cellId, cellInstName, groupId);

    switch (rc) {
```

```

case MWXSuccess:
    MWX_Printf("Successfully Add CellInstance to PlanGroup By name\n");
    return (MWXSuccess);
case MWXFail:
    MWX_Printf("ERROR -- function operater Failed\n");
    return (MWXFail);
case MWXNoLicense:
    MWX_Printf("ERROR -- Failed No License\n");
    return (MWXFail);
default:
    MWX_Printf("ERROR -- unknown return code\n");
    return (MWXFail);
}
}

```

Scheme Equivalent

dbAddCellInstToPlanGroupByName
(see Physical Implementation Online Help)

MWXDb_Add_NetSubType

Add new netSubType to a specific net.

Syntax

```

int MWXDb_Add_NetSubType (
    MWXCellId_t cellId /* INPUT */
    MWXObjId_t netId /* INPUT */
    char *netSubType /* INPUT */
)

```

Parameters

cellId	Database object ID of the cell to which the information applies.
netId	Specifies the database identifier of the net to which the information applies.
netSubType	netSubType that applies to the net. Valid values: "core" "pad" "coreAndPad" "CTS Net" "CTS Root Net" "nonCTS Net"

Return

MWXFail	Occurs if the netSubType for the net is not successfully added.
MWXSuccess	Occurs if the netSubType for the net is successfully added.

Example

```
int mwAddNetSubType(MWXCellId_t cellId,  
                    char *netSubType)  
{  
    int rc;  
    MWX_Printf("INFO -- test MWXDb_Add_NetSubType\n");  
    rc = MWXDb_Add_NetSubType( cellId, netSubType );  
  
    switch (rc) {  
        case MWXSuccess:  
            MWX_Printf("Successfully Add NetSubType\n");  
            return (MWXSuccess);  
        case MWXFail:  
            MWX_Printf("ERROR -- Failed to operate\n");  
            return (MWXFail);  
        case MWXNoLicense:  
            MWX_Printf("ERROR -- Failed No License\n");  
            return (MWXFail);  
        default:  
            MWX_Printf("ERROR -- unknown return code\n");  
            return (MWXFail);  
    }  
}
```

Scheme Equivalent

dbAddNetSubTypeByName

(see Physical Implementation Online Help)

MWXdB_Add_NetSubType_ByName

Adds a net subtype to the net by net name.

Syntax

```
int MWXdB_Add_NetSubType_ByName (
    MWXCellId_t cellId, /* INPUT */
    char *netName, /* INPUT */
    char *netSubType /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell.
netName	Specifies the net name.
netSubType	Specifies the net subtype. Valid values are "core" "pad" "padAndcore" "CTS Net" "CTS Root Net" "nonCTS Net"

Return

MWXFail	Occurs if the subtype cannot be successfully added to the net.
MWXSuccess	Occurs if the subtype is successfully added to the net.

Example

```
int mwAddNetSubTypeByName (MWXCellId_t cellId,
    char *netName,
    char *netSubType)
{
    int rc;

    MWX_Printf("INFO -- test MWXdB_Add_NetSubType_ByName\n");

    rc = MWXdB_Add_NetSubType_ByName ( cellId, netName, netSubType );

    switch (rc) {
        case MWXSuccess:
            MWX_Printf("Successfully Add NetSubType By Name\n");
            return (MWXSuccess);
        case MWXFail:
            MWX_Printf("ERROR -- Failed to operate\n");
            return (MWXFail);
    }
```

```

case MWXNoLicense:
    MWX_Printf("ERROR -- Failed No License\n");
    return (MWXFail);
default:
    MWX_Printf("ERROR -- unknown return code\n");
    return (MWXFail);
}
}

```

Scheme Equivalent

dbAddNetSubTypeByName

(see Physical Implementation Online Help)

MWXDb_Add_Object_ToGroup

Adds an object to a cell group. The cell group is specified by its groupId and the cellId to which it belongs.

Syntax

```

int MWXDb_Add_Object_ToGroup (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t objectId, /* INPUT */
    MWXObjId_t groupId /* INPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell containing the group object.
groupId	Database identifier of the group to which you want to add the object.
objectId	Database identifier of the object that you want to add to the group.

Return

MWXFail	Occurs if the object is not successfully added to the group.
MWXNoLicense	Occurs if the Synopsys license is not present.
MWXSuccess	Occurs if the object is successfully added to the group.

Example

```

int32 mwAddObjectToGroup (MWXCellId_t cellId,
    MWXObjId_t groupId,
    MWXObjId_t objectId)

```

```

{
    int rc;

    MWX_Printf("INFO -- Adding object '%d' to group '%d' ...\n",
objectId, groupId );

    rc = MWXDb_Add_Object_ToGroup(cellId, objectId, groupId);

    switch (rc) {
    case MWXSuccess:
        MWX_Printf("Successfully Add Object to group\n");
        return (MWXSuccess);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to add object\n");
        return (MWXFail);
    case MWXNoLicense:
        MWX_Printf("ERROR -- Failed No License\n");
        return (MWXFail);
    default:
        MWX_Printf("ERROR -- unknown return code\n");
        return (MWXFail);
    }
}

```

Scheme Equivalent

dbAddObjectToGroup

(see Physical Implementation Online Help)

MWXDb_Add_RefLib

Creates a reference library for the opening library.

Syntax

```

int MWXDb_Add_RefLib (
    char *refLibraryName /* INPUT */
)

```

Parameters

refLibraryName Name of the reference library to add to the library.

Return

MWXNoLibraryOpened Occurs if the library is not open.

MWXSuccess Occurs if the reference library is successfully created.

MWXFail Occurs if the reference library cannot be successfully created.

Example

```
int add_ref_lib (
    char *libName,
    char *refLibName)
{
    int rc;

    /*-----
    Open Library
    -----*/
    rc = MWXDb_Open_Lib(libName, &libId);
    if ( rc == MWXSuccess) {
        MWX_Printf ("Successfully opened design Library %s\n", ,libName);
    }
    else {
        MWX_Printf ("ERROR -- Can't open design library\n");
        exit (MWXFail);
    }

    /*-----
    Add_RefLib
    -----*/

    rc = MWXDb_Add_RefLib(reflibName);
    if ( rc == MWXSuccess) {
        MWX_Printf ("Sucessfully added reference library into design library\n");
    }
    else if (rc == MWXNoLibraryOpened) {
        MWX_Printf("Please open the design library first\n");
        exit (MWXFail);
    }
    else {
        MWX_Printf ("ERROR -- Unable to add reference library\n");
        exit(MWXFail);
    }
}
```

```

/*-----
Close the library
-----*/
if(MWXDb_Close_Lib() != MWXSuccess) {
    MWX_Printf("ERROR -- Unable to close design library\n");
    exit(MWXFail);
}
return(MWXSuccess);
}

```

Scheme Equivalent

dbAddRefLib

(see Physical Implementation Online Help)

MWXDb_Archive_CellToLib

Archive a top cell and all its hierarchical references cells from the source library to the destination library. Library unit tiles, cell equivalence and global port tables are always copied if they exist in the source library.

Syntax

```

int MWXDb_Archive_CellToLib(
    char *cellName, /*INPUT*/
    char *arLibName /*INPUT*/
)

```

Parameters

cellName Contains the cell name of the cell to be archived. It may also specify which version of the cell to archive (optional.) It may also specify a new name for the cell in the archive library (optional.)

Syntax: cellName[;versionNumber][=newCellName]
Example: "cellA=cellB" will copy "cellA" from the source library to "cellB" in the destination library.

arLibName Specifies the name of the archive library. If it already exists, then only modified data will be archived to the archived library. If the library doesn't exist, it will be created.

Return

MWXSuccess Occurs if successfully archive cell to lib

MWXFail Occurs if fail to archive cell to library

MWXNoLibraryOpened Occurs if there is no library currently opens

Example

```
int archiveCell (
    char *cellName,
    char *newLibName )
{
    int rc;
    rc = MWXDb_Archive_CellToLib(cellName, newLibName);
    if (rc == MWXFail) {
        MWX_Printf("Fail to archive cell to new library \n");
        return (MWXFail);
    }
    MWX_Printf("Successfully archive cell to new library\n");
    return (MWXSuccess);
}
```

MWXDb_Archive_HierDesign

Archive the current hierarchical design into one new directory.

Syntax

```
int32 MWXDb_Archive_HierDesign(
    char *libName, /*INPUT*/
    char *cellName, /*INPUT*/
    char *arLibName /*INPUT */
)
```

Parameters

libName	Contains the library name of the hierarchical design that you want to archive
cellName	Contains the cell name of the cell to be archived. It may also specify which version of the cell to archive (optional.) It may also specify a new name for the cell in the archive library (optional.) Syntax: cellName[;versionNumber][=newCellName] Example : "cellA=cellB" will copy "cellA" from the source library to "cellB" in the destination library.
arLibName	Specifies the name of the archive library. If it already exists, then only modified data will be archived to the archived library. If the library doesn't exist, it will be created.

Return

MWXSuccess	Occurs if successfully the archive hierarchy design
MWXFail	Occurs if fail to archive the hierarchy design
MWXNoLibraryOpened	Occurs if there is no library currently opens

Example

```
int archive_hierDesign (
    char *libName,
    char *topCellName,
    char *newLibName)
{
    int rc;
    rc = MWXDb_Archive_HierDesign (libName, topCellName, newLibName);
    if (rc == MWXSuccess) {
        MWX_Printf("Successfully archive hierarchical design to new library\n");
        return (MWXSuccess);
    }
    MWX_Printf("Fail to archive hierarchical design to new library \n");
    return (MWXFail);
}
```

MWXDb_Begin_ReadLib

Locks the library. This is useful when performing many operations that involve library read locks as you only need one high level lock rather than having to lock and unlock for each operation.

Syntax

```
int MWXDb_Begin_ReadLib( MWXLibId_t    libId    /* INPUT */ )
```

Parameters

libId	The ID of the library to lock.

Return

MWXSuccess	Occurs if the write lock is successful.
MWXNoLibraryOpened	Occurs if no library is currently opened.
MWXFail	Occurs if the write lock is not successful.

Example

```
int MWXDb_Begin_ReadLib(MWXLibId_t libId) {  
  
    int rc;  
  
    MWX_Printf("INFO -- MWXDb_Begin_ReadLib.\n");  
    rc = MWXDb_Begin_ReadLib(libId);  
  
    switch (rc) {  
    case MWXSuccess:  
        MWX_Printf("Successfully created read lock.\n");  
        return MWXSuccess;  
    case MWXNoLibraryOpened:  
        MWX_Printf("ERROR -- No Library opened. MWXDb_Begin_ReadLib "  
                   "failed.\n");  
        return MWXFail;  
    case MWXFail:  
        MWX_Printf("Unable to create read lock.\n");  
        return MWXFail;  
    default:  
        MWX_Printf("ERROR -- Unknown return code %d.  "  
                   "MWXDB_Begin_ReadLib failed \n",rc);  
        Return MWXFail;  
    }  
}
```

See Also

[MWXDb_End_ReadLib](#)

[MWXDb_Begin_WriteLib](#)

MWXDb_Begin_WriteLib

To begin the library write lock. The update of library lib file is delayed until MWXDb_End_WriteLib is called. This reduces runtime if you successively create many small cells.

Syntax

```
int MWXDb_Begin_WriteLib(  
MWXLibId_t libId /* INPUT */  
)
```

Parameters

libId	Specifies the database identifier of the library to create new cells.

Return

MWXSuccess	Occurs if successfully lock library for write
MWXFail	Occurs if fail to lock library for write
MWXNoLibraryOpened	Occurs if there is no library currently opens

Example

```
MWXDb_Begin_WriteLib (libId);  
for (i=0; i<100; i++) {  
    MWXDb_Create_Cell (cellName, viewName, "Layout", &cellId);  
    ...  
    MWXDb_Close_Cell (cellId);  
}  
  
MWXDb_End_WriteLib (libId);
```

See Also

MWXDb_End_WriteLib

MWXDb_Change_Cell_AccessMode

Changes the current cell access mode from read only to write, or from write to read only.

Syntax

```
int MWXDb_Change_Cell_AccessMode (  
    MWXCellId_t cellId, /* INPUT */  
    char accessMode /* INPUT */  
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of a cell.
<code>accessMode</code>	Specifies the access mode, which can be <code>w</code> or <code>r</code> .

Return

<code>MWXSucces</code>	Occurs if the access mode is successfully changed from read mode to write mode.
<code>MWXFail</code>	Occurs if the assess mode cannot be successfully changed.

Example

```
int mwChangeCellAccessMode (MWXCellId_t cellId, char accessMode)
{
    int rc;

    rc = MWXDb_Change_Cell_AccessMode( cellId, accessMode );
    switch (rc) {
        case MWXSucces:
            MWX_Printf("Successfully Change Cell AccessMode\n");
            return (MWXSucces);
        case MWXFail:
            MWX_Printf("ERROR -- Fail to operate\n");
            return (MWXFail);
        default:
            MWX_Printf("ERROR -- unknown return code\n");
            return (MWXFail);
    }
}
```

Scheme Equivalent

`dbChangeAccessMode`

(see Physical Implementation Online Help)

MWXDb_Check_Cell_Exist

Checks to see if the cell exists in the library or not.

Syntax

```
int MWXDb_Check_Cell_Exist(
    char *cellName /* INPUT */
)
```

Parameters

cellName The name of any cell in the library.

Return

MWXSuccess Occurs if Milkyway is successfully forced to update the cell hierarchy.

MWXFail Occurs if Milkyway is not successfully forced to update cell hierarchy.

Example

```
int mwCheckCellExist (char *libName, char *cellName)
{
    MWXLibId_t libId;
    MWXCellId_t cellId;

    if (MWXDb_Open_Lib(libName,&libId) != MWXSuccess) {
        MWX_Printf("Unable to open library %s\n",libName);
        return(MWXFail);
    }
    else {
        MWX_Printf("Successfully opened library %s\n",libName);
        if (MWXDb_Check_Cell_Exist(cellName) != MWXSuccess) {
            MWX_Printf("Cell %s doesn't exist\n",cellName);
            return (MWXFail);
        } else {
            MWX_Printf("Cell %s existed in library %s\n",cellName, libName);
            if (MWXDb_Open_Cell(cellName,'r",&cellId) != MWXSuccess) {
                MWX_Printf("Fail to open existed cell\n");
                return (MWXFail);
            }
            MWX_Printf("Successfully opened cell %s\n",cellName);
        }
    }
    if (MWXDb_Close_Cell(cellId) != MWXSuccess) {
        MWX_Printf("Unable to close opened cell\n");
        return (MWXFail);
    }
    if (MWXDb_Close_Lib() != MWXSuccess) {
        MWX_Printf("Unable to close library\n");
        return (MWXFail);
    }
    return (MWXSuccess);
}
```

MWXDb_Check_Cell_ExistInLibrary

Check whether the cell exists in the specified library or in any of its reference libraries.

Syntax

```
int32 MWXDb_Check_Cell_ExistInLibrary (  
    MWXLibId_t libId, /* INPUT */  
    char *cellName, /* INPUT */  
    char *viewName /* INPUT */  
)
```

Parameters

libId	Specifies the database identifier of the library.
cellName	Specifies the cell name.
viewName	Specifies the view name.

Return

MWXSuccess	Occurs if the cell exists.
MWXFail	Occurs if the cell does not exist.

Example

```
int mwCheckCellExist (char *libName, char *cellName,  
char *viewName)  
{  
    MWXLibId_t libId;  
    MWXCellId_t cellId;  
  
    if (MWXDb_Open_Lib(libName,&libId) != MWXSuccess) {  
        MWX_Printf("Unable to open library %s\n",libName);  
        return(MWXFail);  
    }  
    else {  
        MWX_Printf("Successfully opened library %s\n",libName);  
        if (MWXDb_Check_Cell_ExistInLibrary(libId,cellName,viewName)  
            != MWXSuccess) {  
            MWX_Printf("Cell %s.%s doesn't exist\n", cellName, viewName);  
            return (MWXFail);  
        } else {  
            MWX_Printf("Cell %s.%s existed in library %s\n",  
                cellName, viewName,libName);  
            if (MWXDb_Open_Cell(cellName,'r',&cellId) != MWXSuccess) {  
                MWX_Printf("Fail to open existed cell\n");  
                return (MWXFail);  
            }  
            MWX_Printf("Successfully opened cell %s\n",cellName);  
        }  
    }  
    if (MWXDb_Close_Cell(cellId) != MWXSuccess) {  
        MWX_Printf("Unable to close opened cell\n");  
        return (MWXFail);  
    }  
    if (MWXDb_Close_Lib() != MWXSuccess) {
```

```

    MWX_Printf("Unable to close library\n");
    return (MWXFail);
}
return (MWXSuccess);
}

```

MWXDb_Check_IdenticalCell

Check for identical CellName between given libraries.

Syntax

```

int MWXDb_Check_IdenticalCell(
char *libName /* INPUT */
)

```

Parameters

libName	Specifies the library name that you want to check for the identical cellName

Return

MWXSuccess	Occurs if successfully check for the identical cellName.
MWXFail	Occurs if fail to check for the identical cellName.
MWXNoLibraryOpened	Occurs if there is no library currently opens.

Example

```

int check_identicalCell_inLib (char *libName) {

    int rc;
    rc = MWXDb_Check_IdenticalCell (libName);
    if (rc == MWXSuccess) {
        MWX_Printf("There are identical cells existed in this library\n");
        return (MWXSuccess);
    }
    MWX_Printf("Fail to check identical cell in this library\n");
    exit (MWXFail);
}

```

MWXDb_Check_RefLibControl

This function checks the reference library control.

Syntax

```
int MWXDb_Check_RefLibControl(  
char *libName /* INPUT */  
)
```

Parameters

libName	Specifies the library name that you want to check for the reference library control

Return

MWXSuccess	Occurs if successfully check for the identical cellName.
MWXFail	Occurs if fail to check for the identical cellName.

Example

```
char *libName = "design";  
int rc;  
rc = MWXDb_Check_RefLibControl (libName);  
switch (rc) {  
    case MWXSuccess:  
        MWX_Printf("Success in checking the reference library control\n");  
        break;  
    case MWXFail:  
        MWX_Printf("Fail to check the reference library control\n");  
        exit(MWXFail);  
    default:  
        MWX_Printf("ERROR -- unknown return code from  
MWXDb_Check_RefLibControl\n");  
        exit(MWXFail);  
}
```

MWXDb_Cleanup_ShapeStruct

Releases a shape data structure of a specified cell. This function should be called after a [MWXDb_Startup_ShapeStruct](#) function call to recover memory allocated for the shape data structure.

Syntax

```
int MWXDb_Cleanup_ShapeStruct(  
    MWXCellId_t cellId/* INPUT */  
)
```

Parameters

`cellId` Specifies the database identifier of the cell to release a shape data structure.

Return

`MWXSuccess` Occurs when the shape data structure is successfully released.

`MWXFail` Occurs when the shape data structure is not released.

Example

See the [MWXDb_Get_Objects_ByBBoxSearch](#) example.

Scheme Equivalent

`dbQuitShapeStruct`
(see Physical Implementation Online Help)

See Also

[MWXDb_Get_Instances_ByBBoxSearch](#)

[MWXDb_Get_Objects_ByBBoxSearch](#)

[MWXDb_Startup_ShapeStruct](#)

MWXDb_Clear_CellMustJoinPin

Removes external connection specifications for pins on a port.

Syntax

```
int MWXDb_Clear_CellMustJoinPin (  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t portId /* INPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell that owns the port and pins.
portId	Specifies the database identifier of the port.

Return

MWXNoLicense	Occurs if no software license is available.
MWXSuccess	Occurs if clear is successful.
MWXFail	Occurs if the must join specification cannot be cleared successfully.

Example

```
int clear_cell_mustJoin_pin (
    MWXCellId_t cellId,
    MWXObjId_t portId
)
{
    int rc;
    rc = MWXDb_Clear_CellMustJoinPin(cellId, portId);
    if (rc == MWXSuccess) {
        MWX_Printf("Successfully clear cellMustJoinPin\n");
        return (MWXSuccess);
    }
    MWX_Printf("Fail to clear cell must join pin\n");
    exit (MWXFail);
}
```

Scheme Equivalent

dbClearMustJoinPortByCell
(see Physical Implementation Online Help)

MWXDb_Clear_RefLibControl

This function clears the reference library control.

Syntax

```
int MWXDb_Clear_RefLibControl(
    char *libName /* INPUT */
)
```


Parameters

libName	Specifies the library name that you want to clear the reference library control

Return

MWXSuccess	Occurs if successfully clear the reference library control.

MWXFail	Occurs if fail to clear the reference library control.

Example

```
char *libName = "design";
int rc;
rc = MWXDb_Clear_RefLibControl (libName);
switch (rc) {
    case MWXSuccess:
        MWX_Printf("Success to clear the reference library control.\n");
        return (MWXSuccess);
    case MWXFail:
        MWX_Printf("Fail to clear the reference library control.\n");
        return (MWXFail);
    default:
        MWX_Printf("ERROR -- unknown return code from
MWXDb_Clear_RefLibControl\n");
        return (MWXFail);
}
```

MWXDb_Close_Cell

Closes the given cell and saves your changes.

Syntax

```
int MWXDb_Close_Cell (
    MWXCellId_t cellId /* INPUT */
)
```

Parameters

`cellId` Specifies the database identifier of the cell to close.

Return

`MWXSuccess` Occurs if the cell is successfully saved and closed.

`MWXFail` Occurs if the cell cannot be closed or is closed without saving changes.

Example

```
/* =====
 * Although, the library automatically saves any open cells when
 * MWXDb_Close_Lib is called, as good practice, always close the cell
 * yourself. This will ensure that your library is clean and temporary
 * files removed.
 * ===== */

int rc;
MWXLibId_t libId;
MWXCellId_t cellId;
char libName[64] = "top_design_library";
char cellName[64] = "top_design.CEL;2"; /* Version 2 of top_design.CEL */

/* Open Library */
rc = MWXDb_Open_Lib(libName, &libId);
if (rc != MWXSuccess && rc != MWXLibraryAlreadyOpened)
    exit(MWXFail);

/* Open Cell */
rc = MWXDb_Open_Cell(cellName, 'w', &cellId);
if (rc != MWXSuccess && rc != MWXCellAlreadyOpened)
    exit(MWXFail);

/* Make changes to the cell */
/* .... */

/* Save Changes and Close Cell */
rc = MWXDb_Close_Cell(cellId);
if (rc != MWXSuccess)
    exit(MWXFail);

MWX_Printf("Successfully closed cell\n");

/* Close Library */
rc = MWXDb_Close_Lib();
if (rc != MWXSuccess)
    exit(MWXFail);
```

Scheme Equivalent

`dbCloseCell` (see *Milkyway Environment Extension Language Reference Manual*)

See Also

[MWXDb_Open_Cell](#)
[MWXDb_Save_Cell](#)
[MWXDb_Close_Cell_And_NoSave](#)
[MWXDb_Close_Cell_And_Save](#)
[MWXDb_Open_Lib](#)
[MWXDb_Open_Library_ByAccessMode](#)
[MWXDb_Close_Lib](#)

MWXDb_Close_Cell_And_NoSave

Closes the given cell and discards your changes.

Syntax

```
int MWXDb_Close_Cell_And_NoSave(  
    MWXCellId_t cellId /* INPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell to close.

Return

MWXSuccess	Occurs if the cell is successfully closed.
MWXNoLibraryOpened	Occurs if a library is not opened.
MWXFail	Occurs if the cell is not closed for any reason.

Example

```
/* =====  
 * Closing a cell and discarding all changes.  
 * ===== */  
  
int rc;  
MWXLibId_t libId;  
MWXCellId_t cellId;  
char libName[64] = "top_design_library";  
char cellName[64] = "top_design.CEL;2"; /* Version 2 of top_design.CEL */  
  
/* Open Library */  
rc = MWXDb_Open_Lib(libName, &libId);  
if (rc != MWXSuccess && rc != MWXLibraryAlreadyOpened)  
    exit(MWXFail);
```

```

/* Open Cell */
rc = MWXDb_Open_Cell(cellName, 'w', &cellId);
if (rc != MWXSuccess && rc != MWXCellAlreadyOpened)
    exit(MWXFail);

/* Make changes to the cell */
/* .... */

/* Discard Changes and Close Cell */
rc = MWXDb_Close_Cell_And_NoSave(cellId);
if (rc != MWXSuccess)
    exit(MWXFail);

MWX_Printf("Successfully closed cell\n");

/* Close Library */
rc = MWXDb_Close_Lib();
if (rc != MWXSuccess)
    exit(MWXFail);

```

See Also

[MWXDb_Open_Lib](#)
[MWXDb_Open_Library_ByAccessMode](#)
[MWXDb_Close_Lib](#)
[MWXDb_Open_Cell](#)
[MWXDb_Close_Cell](#)
[MWXDb_Close_Cell_And_Save](#)

MWXDb_Close_Cell_And_Save

Closes the given cell and saves your changes.

Syntax

```

int MWXDb_Close_Cell_And_Save(
    MWXCellId_t cellId /* INPUT */
)

```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell to close.

Return

MWXSuccess	Occurs if the cell is successfully saved and closed.

MWXNoLibraryOpened	Occurs if a library is not opened.

MWXFail	Occurs if the cell cannot be closed or is closed without saving changes.

Example

```

/* ===== */
*   Although the library automatically saves any open cells when
*   MWXDb_Close_Lib is called, as good practice, always close the cell
*   yourself. This ensures that your library is clean and temporary
*   files are removed.
*   ===== */

int rc;
MWXLibId_t libId;
MWXCellId_t cellId;
char libName[64] = "top_design_library";
char cellName[64] = "top_design.CEL;2"; /* Version 2 of top_design.CEL */

/* Open Library */
rc = MWXDb_Open_Lib(libName, &libId);
if (rc != MWXSuccess && rc != MWXLibraryAlreadyOpened)
    exit(MWXFail);

/* Open Cell */
rc = MWXDb_Open_Cell(cellName, 'w', &cellId);
if (rc != MWXSuccess && rc != MWXCellAlreadyOpened)
    exit(MWXFail);

/* Make changes to the cell */
/* .... */

/* Save Changes and Close Cell */
rc = MWXDb_Close_Cell_And_Save(cellId);
if (rc != MWXSuccess)
    exit(MWXFail);

MWX_Printf("Successfully closed cell\n");
/* Close Library */
rc = MWXDb_Close_Lib();
if (rc != MWXSuccess)
    exit(MWXFail);

```

MWXDb_Close_Cell_And_Save

See Also

[MWXDb_Open_Lib](#)
[MWXDb_Open_Library_ByAccessMode](#)
[MWXDb_Close_Lib](#)
[MWXDb_Open_Cell](#)
[MWXDb_Close_Cell](#)
[MWXDb_Close_Cell_And_NoSave](#)
[MWXDb_Save_Cell](#)

MWXDb_Close_Lib

Closes the currently open design library and its reference libraries. Any open cells from the library are saved and closed.

Syntax

```
int MWXDb_Close_Lib (  
)
```

Parameters

None used.

Return

MWXSuccess	Occurs if the library is successfully closed.
MWXNoLibraryOpened	Occurs if a library is not opened.
MWXFail	Occurs if the library is not closed.

Example

```
int CloseLib () {  
    int rc;  
    MWX_Printf("INFO -- MWXDb_Close_Lib.\n");  
  
    rc = MWXDb_Close_Lib();  
  
    switch (rc) {  
    case MWXSuccess:  
        MWX_Printf("Successfully closed library.\n");  
        return (MWXSuccess);  
    case MWXNoLibraryOpened:  
        MWX_Printf("ERROR -- No library opened.\n");  
        return (MWXFail);  
    case MWXFail:  
        MWX_Printf("ERROR -- Library failed to close.\n");  
        return (MWXFail);  
    default:  
        MWX_Printf("UNKNOWN ERROR %d from MWXDb_Close_Lib.\n", rc);  
    }
```

```
    return (MWXFail);  
}  
}
```

Scheme Equivalent

dbCloseLib
(see Physical Implementation Online Help)

See Also

[MWXDb_Open_Lib](#)
[MWXDb_Open_Library_ByAccessMode](#)
[MWXDb_Open_Cell](#)
[MWXDb_Close_Cell](#)
[MWXDb_Close_Cell_And_Save](#)
[MWXDb_Close_Cell_And_NoSave](#)

MWXDb_Connect

Connects the object identified by `objectId` to the net identified by `netId`. The only object types allowed are ports, port instances, pins, wires, contacts, contact arrays, paths, rectangles, and polygons.

Syntax

```
int MWXDb_Connect(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t netId, /* INPUT */  
    MWXObjId_t objectId /* INPUT */  
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell containing the net and the object to connect.
<code>netId</code>	Specifies the database identifier of the net to which the object is to connect.
<code>objectId</code>	Specifies the database identifier of the object to connect to the net.

Return

<code>MWXSuccess</code>	Occurs if the object is successfully connected to the net.
<code>MWXFail</code>	Occurs if the object is not successfully connected to the net.

Example

```
int connect_object_to_net (
    MWXCellId_t cellId,
    MWXObjId_t netId,
    MWXObjId_t objectId
)
{
    if (MWXDb_Connect (cellId, netId, objId) != MWXSuccess) {
        MWX_Printf("Fail to connect object to net\n");
        return (MWXFail);
    }
    MWX_Printf("Succesffully connect object to net\n");
    return (MWXSuccess);
}
```

Scheme Equivalent

`dbConnect`

(see Physical Implementation Online Help)

MWXDb_Connect_PortInstance_ByName

Connects the specified port instance to the net identified by `netId`.

Syntax

```
int MWXDb_Connect_PortInstance_ByName (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t netId, /* INPUT */
    char *cellInstanceName, /* INPUT */
    char *portInstanceName /* INPUT */
)
```


Parameters

cellId	Specifies the database identifier of the cell containing the net and the cell instance owning the port instance to connect.
netId	Specifies the database identifier of the net to which the port instance is to connect.
cellInstanceName	Specifies the name of the cell instance containing the port instance to connect.
portInstanceName	Specifies the name of the port instance to connect.

Return

MWXSuccess	Occurs if the port instance is successfully connected to the net.
MWXFail	Occurs if the port instance is not successfully connected to the net.

Example

```
int connect_portInst_to_net (
    MWXCellId_t cellId,
    MWXObjId_t netId,
    char *cellInstanceName,
    char *portInstanceName
)
{
    if (MWXDb_Connect_PortInstance_ByName (cellId, netId, cellInstanceName,
    portInstanceName) != MWXSuccess) {
        MWX_Printf("Fail to connect portInstance by name\n");
        return (MWXFail);
    }
    MWX_Printf("Successfully connect portInstance by name \n");
    return (MWXSuccess);
}
```

Scheme Equivalent

dbConnectPortInstByName
(see Physical Implementation Online Help)

MWXDb_Connect_PortWithPin

Connects the specified port with a pin.

Syntax

```
int32 MWXDb_Connect_PortWithPin (
    MWXCellId_t cellId /*INPUT*/
    MWXObjId_t  netId  /*INPUT*/
    MWXObjId_t  portId /*INPUT*/
)
```

Parameters

cellId	Specifies the database identifier of the cell containing the net to which the port is connected.
netId	Specifies the database identifier of the net to which the port is connected.
portId	Specifies the database identifier of the port to which the pin will be connected.

Return

MWXSuccess	Occurs if the port is successfully connected with the pin.
MWXFail	Occurs if the port fails to connect with the pin.

Example

```
int connect_port_to_net_with_pin (
    MWXCellId_t cellId,
    MWXObjId_t netId,
    MWXObjId_t portId)
{
    if (MWXDb_Connect_PortWithPin(cellId, netId, portId) != MWXSuccess) {
        MWX_Printf("Fail to connect port with pin\n");
        return (MWXFail);
    }
    MWX_Printf("Successfully connect port with pin\n");
    return (MWXSuccess);
}
```

MWXDb_Convert_ObjectsToPath

Converts the database objects of `objId` that are contained in the cell identified by `cellId` to the path objects. Wires, rectangles, and rectangular polygons are legal objects to convert.

Syntax

```
int MWXDb_Convert_ObjectsToPath(
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t objId, /* INPUT */
    MWXObjId_t *pathId /* OUTPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell containing objects to convert.
objId	Specifies the type name (from mwX.h) of the objects to convert.
pathId	Specifies the database identifier of the returned path objects.

Return

MWXNoLicense	Occurs if the objects are successfully converted.
MWXSuccess	Occurs if the objects are not successfully converted.

Example

```
int rc;
MWXPoint pointArr[2];

// Create a rectangle
pointArr[0].x = 500; pointArr[0].y = 500;
pointArr[1].x = 600; pointArr[1].y = 600;

rc = MWXDb_Create_Rectangle(cellId, "M1R", pointArr, &objId);
if (rc != MWXSuccess ) {
    MWX_Printf("*** ERROR - Failed to create rectangle.\n");
    exit(MWXFail);
}

MWX_Printf("Converting database object to path\n");
rc = MWXDb_Convert_ObjectsToPath(cellId objId, &pathId);

switch (rc) {
case MWXSuccess:
    MWX_Printf("SUCCESS : Object converted to path(%d)\n", pathId);
    break;
case MWXFail:
    MWX_Printf("*** ERROR : Failed to convert object to path.\n");
    exit(MWXFail);
default:
    MWX_Printf("*** UNKNOWN ERROR %d from MWXDb_Convert_ObjectsToPath.\n", rc);
    exit(MWXFail);
}
```

MWXDb_Convert_PathToPolygon

Converts the path identified by `pathId` in the specified cell to a polygon, and returns the object identifier of the new polygon.

Syntax

```
int MWXDb_Convert_RectangleToPolygon(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t pathId, /* INPUT */  
    MWXObjId_t *polyId /* OUTPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell containing the path to convert.
pathId	Specifies the database identifier of the path to convert.
polyId	Returns the database identifier of the resulting polygon.

Return

MWXSuccess	Occurs if the path is successfully converted.
MWXFail	Occurs if the path is not successfully converted.

Example

```
int ConvertPathToPolygon(MWXCellId_t cellId, MWXObjId_t *polyId) {  
  
    int rc;  
    MWXObjId_t pathId;  
    MWXPoint_t pointArray[64];  
  
    MWX_Printf("INFO -- MWXDb_Create_Path.\n");  
    /* Specify the points for the path */  
    pointArray[0].x = 400; pointArray[0].y = 400;  
    pointArray[1].x = 400; pointArray[1].y = 0;  
    pointArray[2].x = 0; pointArray[2].y = 0;  
    pointArray[3].x = 0; pointArray[3].y = 400;  
  
    /* Create a new path on layer METAL5 (defined in tech file) of this cell  
       with the first 4 points of pointArray */  
    rc = MWXDb_Create_Path(cellId, "METAL5", 0, 10, 0, 1, 4,  
        pointArray, &pathId);  
  
    if (rc == MWXSuccess) {  
        MWX_Printf("Successfully created path(%d).\n", pathId);  
    } else {  
        MWX_Printf("ERROR -- Function MWXDb_Create_Path failed.\n");  
        return(MWXFail);  
    }  
  
    /* Convert the newly created path into a polygon. If the  
       conversion is successful, the new polygon's object ID is polyId */  
    MWX_Printf("INFO -- MWXDb_Convert_PathToPolygon.\n");  
    rc = MWXDb_Convert_PathToPolygon(cellId, pathId, polyId);  
}
```

```

switch (rc) {
case MWXSuccess:
MWX_Printf("Successful path => polygon conversion. Polygon Id = %d.\n",
          *polyId);
break;
case MWXFail:
MWX_Printf("ERROR -- MWXDb_Convert_PathToPolygon failed.\n");
return(MWXFail);
default:
MWX_Printf("UNKNOWN ERROR %d from MWXDb_Convert_PathToPolygon.", rc);
return(MWXFail);
}

return(MWXSuccess);
}

```

See Also

[Path Object](#) and Field Descriptions
[Polygon Object](#) and Field Descriptions
[MWXDb_Convert_ObjectsToPath](#)
[MWXDb_Convert_PathToRectangles](#)
[MWXDb_Create_Path](#)
[MWXDb_Create_Polygon](#)

MWXDb_Convert_PathToRectangles

Converts the path identified by `pathId` in the specified cell into one or more rectangles whose object identifiers are returned in `rectIdArray`. It is your responsibility to free the memory of the returned array.

Syntax

```

int MWXDb_Convert_PathToRectangles(
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t pathId, /* INPUT */
    MWXObjId_t **rectIdArray /* OUTPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell containing the path to convert.
pathId	Specifies the database identifier of the path to convert.
rectIdArray	Returns an array containing the resulting rectangle IDs.

Return

MWXSuccess	Occurs if the path is successfully converted.
MWXFail	Occurs if the path is not successfully converted.

Example

```
int ConvertPathToRectangles(MWXCellId_t cellId, MWXObjId_t **rectIdArray) {  
  
    int rc, i;  
    MWXObjId_t pathId;  
    MWXPoint_t pointArray[64];  
  
    MWX_Printf("INFO -- MWXDb_Create_Path.\n");  
  
    /* Specify the points for the path */  
    pointArray[0].x = 400; pointArray[0].y = 400;  
    pointArray[1].x = 400; pointArray[1].y = 0;  
    pointArray[2].x = 0; pointArray[2].y = 0;  
    pointArray[3].x = 0; pointArray[3].y = 400;  
  
    /* Create a new path on layer METAL5 (defined in tech file) of this cell  
    with the first 4 points of pointArray */  
    rc = MWXDb_Create_Path(cellId, "METAL5", 0, 10, 0, 1, 4, pointArray, &pathId);  
  
    if (rc == MWXSuccess) {  
        MWX_Printf("Successfully created path(%d).\n", pathId);  
    } else {  
        MWX_Printf("ERROR -- Function MWXDb_Create_Path failed\n");  
        return(MWXFail);  
    }  
  
    /* Convert the newly created path into rectangles. If the  
    conversion is successful, the array of rectangles is referenced by  
    rectIdArray */  
    MWX_Printf("INFO -- MWXDb_Convert_PathToRectangles.\n");  
    rc = MWXDb_Convert_PathToRectangles(cellId, pathId, rectIdArray);  
  
    switch (rc) {  
        case MWXSuccess:  
            MWX_Printf("Path converted to rectangles.\n");  
  
            for (i = 0; *(*rectIdArray + i) != 0; i++) {  
                MWX_Printf("rectIdArray[%d] = %d\n", i, *(*rectIdArray + i));  
            }  
        }  
}
```

```

}

return(MWXSuccess);
case MWXFail:
MWX_Printf("ERROR -- Function MWXDb_ConvertPathToRectangles failed.\n");
return(MWXFail);
default:
MWX_Printf("UNKNOWN ERROR %d from MWXDb_Convert_PathToRectangles.", rc);
return(MWXFail);
}

}

```

See Also

[Path Object](#) and Field Descriptions
[Rectangle Object](#) and Field Descriptions
[MWXDb_Convert_ObjectsToPath](#)
[MWXDb_Convert_PathToRectangles](#)
[MWXDb_Create_Path](#)
[MWXDb_Create_Rectangle](#)

MWXDb_Convert_Point_ToDBPoint

Converts a coordinate pair from user units to database units. This function does not perform the division with the `lengthPrecision` value defined in the technology data used in the library. You must perform this step.

Syntax

```

int MWXDb_Convert_Point_ToDBPoint (
    MWXPoint_t *point, /* INPUT */
    MWXPoint_t *DBpoint /* OUTPUT */
)

```

Parameters

<code>point</code>	Specifies the coordinate pair, in user units.
<code>DBpoint</code>	Specifies the coordinate pair, in database units.

Return

<code>MWXNoLicense</code>	Occurs if no software license is available.
<code>MWXSuccess</code>	Occurs if the conversion is successful.

Example

```

length_precision = 1000.0;
rc = MWXDb_Convert_Point_ToDBPoint (&point,&DBpoint);

```

```

    if (rc == MWXSuccess) {
        X = DBpoint.x / length_precision;
        Y = DBpoint.y / length_precision;
    }

```

MWXDb_Convert_PolygonToRectangles

Converts the polygon identified by `polyId` in the specified cell into one or more rectangles whose object identifiers are returned in `rectIdArray`. If the polygon was originally rectangular, the result of the conversion will be one rectangle. Otherwise, one polygon can be converted into many rectangles. It is your responsibility to free the memory of the returned array.

Syntax

```

int MWXDb_Convert_PolygonToRectangles(
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t polyId, /* INPUT */
    MWXObjId_t **rectArrayId /* OUTPUT */
)

```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell containing the polygon to convert.
<code>polyId</code>	Specifies the database identifier of the polygon to convert.
<code>rectArrayId</code>	Returns an array containing the resulting rectangle IDs.

Return

<code>MWXSuccess</code>	Occurs if the polygons are successfully converted.
<code>MWXFail</code>	Occurs if the polygons are not successfully converted.

Example

```

int ConvertPolygonToRectangles(MWXCellId_t cellId, MWXObjId_t **rectIdArray) {
    int rc, i;
    MWXObjId_t polyId;
    MWXPoint_t pointArray[64];

    /* Specify the points for the polygon */
    pointArray[0].x = 0; pointArray[0].y = 0;
    pointArray[1].x = 400; pointArray[1].y = 0;
    pointArray[2].x = 400; pointArray[2].y = 400;
    pointArray[3].x = 800; pointArray[3].y = 400;
    pointArray[4].x = 800; pointArray[4].y = 0;
    pointArray[5].x = 1000; pointArray[5].y = 0;

```



```

    pointArray[6].x = 1000; pointArray[6].y = 800;
    pointArray[7].x = 0; pointArray[7].y = 800;

/* Create a new polygon on layer METAL5 (defined in tech file) of this cell */
MWX_Printf("INFO -- MWXDb_Create_Polygon.\n");
rc = MWXDb_Create_Polygon(cellId, "METAL5", 8, pointArray, &polyId);

if (rc == MWXSuccess) {
MWX_Printf("Successfully created polygon (%d)\n",polyId);
} else {
MWX_Printf("ERROR - Function MWXDb_Create_Polygon failed\n");
return (MWXFail);
}

MWX_Printf("INFO -- MWXDb_Convert_PolygonToRectangles.\n");
rc = MWXDb_Convert_PolygonToRectangles(cellId, polyId, rectIdArray);

switch (rc) {
case MWXSuccess:
MWX_Printf("Successful Polygon => Rectangle conversion\n");

for (i = 0; *(*rectIdArray + i) != 0; i++) {
MWX_Printf("rectIdArray[%d] = %d\n", i, *(*rectIdArray + i));
}

return (MWXSuccess);
case MWXFail:
MWX_Printf("ERROR -- Function MWXDb_Convert_PolygonToRectangles
          failed.\n");
return (MWXFail);
default:
MWX_Printf("UNKNOWN ERROR %d from MWXDb_Convert_PolygonToRectangles.\n",
          rc);
return (MWXFail);
}
}

```

See Also

[Polygon Object and Field Descriptions](#)
[Rectangle Object and Field Descriptions](#)
[MWXDb_Convert_ObjectsToPath](#)
[MWXDb_Convert_PathToPolygon](#)
[MWXDb_Convert_RectangleToPolygon](#)
[MWXDb_Create_Polygon](#)
[MWXDb_Create_Rectangle](#)

MWXDb_Convert_RectangleToPolygon

Converts the rectangle identified by `rectId` in the specified cell into a polygon.

Syntax

```
int MWXDb_Convert_RectangleToPolygon(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t rectId, /* INPUT */  
    MWXObjId_t *polyId /* OUTPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell containing the rectangle to convert.
rectId	Specifies the database identifier of the rectangle to convert.
polyId	Returns the database identifier of the resulting polygon.

Return

MWXSuccess	Occurs if the rectangle is successfully converted.
MWXFail	Occurs if the rectangle is not successfully converted.

Example

```
int ConvertRectangleToPolygon(MWXCellId_t cellId, MWXObjId_t *polyId) {  
  
    int rc;  
    MWXObjId_t rectId;  
    MWXPoint_t pointArray[64];  
  
    // Specify the points for the rectangle  
    pointArray[0].x = 400; pointArray[0].y = 400;  
    pointArray[1].x = -100; pointArray[1].y = -300;  
  
    // Create a new rectangle on layer METAL5 (defined in tech file) of this cell  
    // The first two points of pointArray are used to create the rectangle  
    MWX_Printf("INFO -- MWXDb_Create_Rectangle.\n");  
    rc = MWXDb_Create_Rectangle(cellId, "METAL5", pointArray, &rectId);  
  
    if (rc == MWXSuccess) {  
        MWX_Printf("Successfully created rectangle(%d).\n", rectId);  
    } else {  
        MWX_Printf("ERROR -- Function MWXDb_Create_Rectangle failed\n");  
        return(MWXFail);  
    }  
  
    MWX_Printf("INFO -- MWXDb_Convert_RectangleToPolygon.\n");  
    rc = MWXDb_Convert_RectangleToPolygon(cellId, rectId, polyId);  
  
    if (rc == MWXSuccess) {  
        MWX_Printf("Successful Rectangle => Polygon conversion. Polygon ID = %d\n", *polyId);  
    } else {  
        MWX_Printf("ERROR -- Function MWXDb_Convert_RectangleToPolygon failed\n");  
        return(MWXFail);  
    }  
}
```

```

}

return (MWXSuccess);
}

```

See Also

[Rectangle Object and Field Descriptions](#)
[Polygon Object and Field Descriptions](#)
[MWXDb_Convert_PolygonToRectangles](#)
[MWXDb_Convert_PathToRectangles](#)
[MWXDb_Convert_ObjectsToPath](#)
[MWXDb_Create_Rectangle](#)
[MWXDb_Create_Polygon](#)

MWXDb_Copy_Cell

Creates a copy of the cell specified by `cellName1` and names it `cellName2`. The original cell (`cellName1`) must be in the currently open design library. If the copy is successful, `cellName2` is placed in the design library and opened in write mode. The copy cell operation also copies cell attach files, but it is up to you to copy any additional cell views.

Note:

The default cell version is the largest existing version in the specified view. The default cell view is CEL if no cell view is specified.

Syntax

```

int MWXDb_Copy_Cell (
    char *cellName1, /* INPUT */
    char *cellName2, /* INPUT */
    MWXCellId_t *cellId /* OUTPUT */
)

```

Parameters

cellName1	Specifies the name of the cell to copy. Syntax: cellName[.viewName][:version]
cellName2	Specifies the name of the new cell. Syntax: cellName[.viewName] The destination cell version is the largest version of the cell in the given view + 1
cellId	Specifies the database ID of the new cell.

Return

MWXSuccess	Occurs if the cell is successfully copied.
MWXNoLibraryOpened	Occurs if a library is not open.
MWXFail	Occurs if the copy fails for any other reason.

Example

```
/*=====
 * It is sometimes useful to make a copy of a cell, for example, so that you
 * can test and/or modify a copy instead of the original.
 *=====*/

int rc;
MWXLibId_t libId;
MWXCellId_t cellId;
char libName[64] = "top_design_library";
char cellName1[64] = "top_design.CEL;2"; /* Version 2 of top_design.CEL */
char cellName2[64] = "copy_of_top_design.CEL";

/* Open Library */
rc = MWXDb_Open_Lib(libName, &libId);
if (rc != MWXSuccess && rc != MWXLibraryAlreadyOpened)
    exit(1);

/* Copy Cell from cellName1 to cellName2*/
rc = MWXDb_Copy_Cell(cellName1, cellName2, &cellId);
if (rc != MWXSuccess)
    exit(1);
else
    MWX_Printf("Successfully copied cell from %s => %s with cellId = %d\n",
cellName1, cellName2, cellId);

/* Make changes to the cell */
/* .... */

/* Save Changes and Close Cell */
rc = MWXDb_Close_Cell(cellId);
if (rc != MWXSuccess)
```

```
exit(1);

/* Close Library */
rc = MWXDb_Close_Lib();
if (rc != MWXSuccess)
    exit(1);
```

Scheme Equivalent

dbCopyCell
(see Physical Implementation Online Help)

See Also

[MWXDb_Copy_CellFromAnotherLib](#)
[MWXDb_Copy_CellToAnotherLibDisk](#)
[MWXDb_Copy_CellWithTIM](#)
[MWXDb_Copy_Lib](#)
[MWXDb_Create_Cell](#)
[MWXDb_Delete_Cell](#)

MWXDb_Copy_CellFromAnotherLib

Creates a copy of the cell specified by `cellName1` and names it `cellName2` with the same functionality as the `MWXDb_Copy_Cell` function, except that the cell is copied from another library to the current library.

Syntax

```
int MWXDb_Copy_CellFromAnotherLib (
    char *srcCellName, /* INPUT */
    char *srcLibName, /* INPUT */
    char *dstCellName, /* INPUT */
    MWXCellId_t *cellId /* OUTPUT */
)
```

Parameters

srcCellName	Specifies the name of the cell from the source library to copy. Syntax: cellName[.viewName][:version]
srcLibName	Specifies the source library name.
dstCellName	Specifies the name of the new cell. Syntax: cellName[.viewName] The destination cell version will always be the largest version of the cell in the given view + 1.
cellId	Specifies the database ID of the new cell.

Return

MWXNoLibraryOpened	Occurs if the library is not open.
MWXSuccess	Occurs if the cell is successfully copied from any other library resource.
MWXFail	Occurs if the cell cannot be successfully copied.

Example

```
/*=====
=====
 * Sometimes you may wish to have a copy of a cell in the current library instead
of a reference library. This way,
 * you can make modifications to the cell without changing the original cell
or having to close the current
 * library.
*=====
=====*/

int rc;
MWXLibId_t libId;
MWXCellId_t cellId;
char libName[64] = "top_design_library";

char srcCellName[64] = "nand_macro.CEL";
char srcLibName[64] = "macro_cell_library";
char dstCellName[64] = "nand_macro.CEL";

/* Open Library */
rc = MWXDb_Open_Lib(libName, &libId);
if (rc != MWXSuccess && rc != MWXLibraryAlreadyOpened)
    exit(1);

/* Copy Cell from srcCellName, srcLibName to dstCellName*/
```

```

    rc = MWXDb_Copy_CellFromAnotherLib(srcCellName, srcLibName, dstCellName,
&cellId);
    if (rc != MWXSuccess)
        exit(1);
    else
        MWX_Printf("Successfully copied cell from %s/%s => %s with cellId = %d\n",
srcLibName, srcCellName, dstCellName, cellId);

/* Open Cell */
rc = MWXDb_Open_Cell(dstCellName, 'w', &cellId);
if (rc != MWXSuccess && rc != MWXCellAlreadyOpened)
    exit(1);

/* Make changes to the cell */
/* .... */

/* Save Changes and Close Cell */
rc = MWXDb_Close_Cell(cellId);
if (rc != MWXSuccess)
    exit(1);

/* Close Library */
rc = MWXDb_Close_Lib();
if (rc != MWXSuccess)
    exit(1);

```

Scheme Equivalent

dbCopyCellFromAnotherLib
(see Physical Implementation Online Help)

See Also

[MWXDb_Copy_Cell](#)
[MWXDb_Copy_CellToAnotherLibDisk](#)
[MWXDb_Copy_CellWithTIM](#)
[MWXDb_Copy_Lib](#)

MWXDb_Copy_CellToAnotherLibDisk

Copies a cell to another library disk and returns the database identifier of the new cell.

Syntax

```

int MWXDb_Copy_CellToAnotherLibDisk(
char *cellName1, /* INPUT */
char *libName1, /* INPUT */
char *cellName2, /* INPUT */
MWXCellId_t *cellId /* OUTPUT-The cellId of cellName2 */
)

```

Parameters

cellName1	Specifies the cell name of the cell that you want to copy to another lib
libName1	Specifies the name of the library that you want to copy the cell to
cellName2	Specifies the new name for the cell that you want to copy in the destination library
cellId	Specifies the cellId of the new cell in the destination library

Return

MWXSuccess	Occurs if successfully copy cell to another library disk.
MWXFail	Occurs if fail to copy cell to another library disk.
MWXNoLibraryOpened	Occurs if there is no library currently opens.

Example

```
char *libName = "design";
char *cellName1 = "toprt";
char *cellName2 = "new";
MWXCellId_t *cellId;
int rc;
rc = MWXDb_Copy_CellToAnotherLibDisk (cellName1, libName, cellName2, &cellId);
switch (rc) {
    case MWXSuccess:
        MWX_Printf("Successfully copy cell to another library disk. New cellId
= %d.\n", cellId);
        break;
    case MWXFail:
        MWX_Printf("Fail to copy cell to another library disk.\n");
        break;
    case MWXNoLibraryOpened:
        MWX_Printf("Library no open\n");
        break;
    default:
        MWX_Printf("ERROR -- unknown return code from
MWXDb_Copy_CellToAnotherLibDisk\n");
        break;
}
```

MWXDb_Copy_CellWithTIM

Copies a cell from the same library, with its timing view.

Syntax

```
int MWXDb_Copy_CellWithTIM (
    char *cellName1, /* INPUT */
    char *cellName2, /* INPUT */
    MWXCellId_t *cellId /* OUTPUT */
    /* The opened cell ID of cellName2. */
)
```

Parameters

cellName1	Specifies the name of the cell to copy.
cellName2	Specifies the name of the cell to duplicate from the original.
cellId	The database identifier of the new cell.

Return

MWXNoLibraryOpened	Occurs if the library is not open.
MWXSuccess	Occurs if the cell copies with timing information and the successful function retrieves the database identifier of the cell.
MWXFail	Occurs if the cell cannot be successfully copied with timing information.

Example

```
rc = MWXDb_Copy_CellWithTIM ( cellName1, cellName2, &cellId );
switch (rc) {
    case MWXSuccess:
        return (MWXSuccess)
    case MWXFail:
        MWX_Printf ("ERROR -- Failed to operate. \n");
        return (MWXFail);
}
```

MWXDb_Copy_Lib

Creates a duplicate of the library specified by `libraryName1` and gives it the name specified by `libraryName2`. The library being copied must be closed before the function is called.

Syntax

```
int MWXDb_Copy_Lib (  
    char *libraryName1, /* INPUT */  
    char *libraryName2 /* INPUT */  
)
```

Parameters

libraryName1	Specifies the name of the library to copy.
libraryName2	Specifies the name to give to the duplicate library.

Return

MWXSuccess	Occurs if the copy is successful.
MWXLibraryAlreadyOpened	Occurs if the library being copied is open.
MWXFail	Occurs if the copy fails for any reason.

Example

```
int CopyLib (char *libraryName1, char *libraryName2) {  
  
    int rc;  
    MWX_Printf("INFO -- MWXDb_Copy_Lib.\n");  
  
    rc = MWXDb_Copy_Lib(libraryName1, libraryName2);  
    switch (rc) {  
    case MWXSuccess:  
        MWX_Printf("Successfully copied contents of library %s to library %s.\n",  
            libraryName1, libraryName2);  
        return (MWXSuccess);  
    case MWXLibraryAlreadyOpened:  
        MWX_Printf("ERROR -- LIBRARY ALREADY OPENED. Function MWXDb_Copy_Lib  
            failed.\n");  
        return (MWXFail);  
    case MWXFail:  
        MWX_Printf("ERROR -- Function MWXDb_Copy_Lib failed.\n");  
        return (MWXFail);  
    default:  
        MWX_Printf("UNKNOWN ERROR %d from MWXDb_Copy_Lib.\n", rc);  
        return (MWXFail);  
    }  
}
```

Scheme Equivalent

dbCreateLib
(see Physical Implementation Online Help)

See Also

[MWXDb_Create_Lib](#)
[MWXDb_Copy_Cell](#)
[MWXDb_Copy_CellFromAnotherLib](#)
[MWXDb_Copy_CellToAnotherLibDisk](#)
[MWXDb_Copy_CellWithTIM](#)

MWXDb_Create_Cell

Creates an empty cell in write mode with the view name and type supplied by `viewName` and `viewType`. The cell is given the name specified by `cellName`. A library must be open for a new cell to be created. Note that if the command successfully completes, the cell will be open.

Syntax

```
int MWXDb_Create_Cell (
    char *cellName, /* INPUT */
    char *viewName, /* INPUT */
    char *viewType, /* INPUT */
    MWXCellId_t *cellId /* OUTPUT */
)
```

Parameters

<code>cellName</code>	Specifies the name of cell to create.
<code>viewName</code>	Specifies the name of the view to give to the cell (valid values are .cel, .fram, .exp, .netl, and .tim).
<code>viewType</code>	Specifies the type of view (valid values are Layout, Netlist, Unknown).
<code>cellId</code>	Specifies the database ID of the new cell.

Return

<code>MWXSuccess</code>	Occurs if the cell is successfully created.
<code>MWXNoLibraryOpened</code>	Occurs if a library is not open.
<code>MWXFail</code>	Occurs if the cell creation fails for any other reason.

Example

```
/*-----
 * creating cell from scratch
 *-----*/
```

```

MWXCellId_t cellId;
char *cellName = "CELL1";
char *viewName = ".CEL";
char *viewType = "Netlist";

rc = MWXDb_Create_Cell(cellName, viewName, viewType, &cellId);
if (rc == MWXFail) {
    MWX_Printf("Failed to create cell(%s) viewName(%s) viewType(%s)\n",
               cellName, viewName, viewType);
    exit(MWXFail);
}
else if (rc == MWXNoLibraryOpened) {
    MWX_Printf("No library is open\n");
}
else {
    MWXCellId_t openCellId = 0;
    MWX_Printf("Succeeded to create cell(%s)\n", cellName);
    MWX_Printf("Try to open cell(%s)\n", cellName);
    rc = MWXDb_Open_Cell(cellName, 'r', &openCellId); /* NOTE 'r' mode would
prevent changes! */
    if (rc == MWXCellAlreadyOpened) {
        MWX_Printf("Cell is already open\n");
    }
    else if (rc == MWXNoLibraryOpened) {
        MWX_Printf("No library is open\n");
    }
    else if (rc == MWXSuccess) {
        MWX_Printf("Succeeded to open cell\n");
    }
    else {
        MWX_Printf("Failed to open cell(%s)\n", cellName);
    }
}
}

```

Scheme Equivalent

dbDeleteCell

(see Physical Implementation Online Help)

MWXDb_Create_Cell_Property

Creates a property with the name given by `propertyName` and assigns the value given by `propertyValue`. Note that `propertyId` will be undefined if the return code is not `MWXSuccess`.

Syntax

```

int MWXDb_Create_Cell_Property(
    MWXCellId_t cellId, /* INPUT */
    char *propertyName, /* INPUT */
    unsigned char propertyDataType, /* INPUT */
    MWXPropValue_t *propertyValue, /* INPUT */
    uint16 propertySize, /* INPUT */
    MWXObjId_t *propertyId /* OUTPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell in which to create the property.
propertyName	Specifies the name of the property to create.
propertyDataType	Specifies the type of property to create; legal values can be found in the mwX.h file: MWXcPropBooleanTypeMWXcPropIntegerTypeMWXcPropFloatingTypeMWXcPropStringTypeMWXcPropDataStructTypeMWXcPropTimeType
propertyValue	Specifies the value to assign to the new property.
propertySize	Specifies the memory size, in bytes, of the property value (for string, use string length, for all others, use size of data type).
propertyId	Specifies the database identifier of the new property.

Return

MWXSuccess	Occurs if the property is successfully created.
MWXFail	Occurs if the property is not successfully created.

Example

```
MWXObjId_t create_cell_property (  
    MWXCellId_t cellId,  
    char *propertyName,  
    unsigned char propertyDataType,  
    MWXPropValue_t *propertyValue)  
{  
    MWXPropValue_t *propertyValue;  
    int rc;  
    uint16 propertySize;  
    MWXObjId_t propertyId;  
  
    switch (propertyDataType) {  
        case MWXcPropBooleanType:  
            propertySize = sizeof (propertyValue.pBoolean);  
            break;  
        case MWXcPropIntegerType:  
            propertySize = sizeof (propertyValue.pInt);  
            break;  
        case MWXcPropFloatingType:  
            propertySize = sizeof (propertyValue.pFloat);  
            break;  
        case MWXcPropStringType:  
            propertySize = sizeof (propertyValue.pString);  
    }
```

```

        break;
    default:
        exit (MWXFail);
    }
    rc = MWXDb_Create_Cell_Property (cellId,  propertyName, propertyDataType,
    &propertyValue, propertySize, &propertyId);
    if (rc != MWXSuccess) {
        MWX_Printf("Fail to create cell property \n");
        exit (MWXFail);
    }
    MWX_Printf("Successfully create cell property\n");
    return (propertyId);
}

```

Scheme Equivalent

dbCreateCellProperty

(see Physical Implementation Online Help)

MWXDb_Create_CellAttachFile

Creates a UNIX file attached to an open cell. This file is writable after creation. It is your responsibility to maintain the attached file after its creation.

Syntax

```

FILE * MWXDb_Create_CellAttachFile (
    MWXCellId_t cellId, /* INPUT */
    char *pseudoAccessName, /* INPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell to which a created file is attached.
pseudoAccessName	Identifies the attached file for future access. It must be unique in a cell.

Return

FILE *	A pointer of identifier <i>FILE</i> , defined in <i>stdio.h</i> , is returned when the attached file is successfully created.
NULL(0)	Occurs if an attached file is not created.

Example

```

int create_cellAttachedFile (
    MWXCellId_t cellId,
    char *pseudoAccessName )

```

```

{
    FILE* fp;
    fp = MWXDb_Create_CellAttachFile(cellId, pseudoAccessName);
    if (fp != NULL) {
        MWX_Printf("INFO -- cell attached file with the pseudoAccessName = %s
is created\n",pseudoAccessName);
        return (MWXSuccess);
    }
    else {
        MWX_Printf("ERROR -- Failed to create cellAttached file\n");
        return (MWXFail);
    }
}

```

MWXDb_Create_CellBoundary

Creates a cell boundary in the cell.

Syntax

```

int MWXDb_Create_CellBoundary (
    MWXCellId_t cellId, /* INPUT */
    int32 numberOfPoint, /* INPUT */
    MWXPoint_t pointArray[] /* INPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell.
numberOfPoint	Number of points.
pointArray	An array of the coordinates.

Return

MWXNoLibraryOpened	Occurs if the library is not open.
MWXFail	Occurs if the boundary is not successfully created.
MWXSuccess	Occurs if the boundary is successfully created.

Example

```

int create_cellboundary (
    MWXCellId_t cellId)
{
    int32 numberOfPoint = 4;
    MWXPoint_t pointArray[4];

    pointArray[0].x = 400; pointArray[0].y = 400;
    pointArray[1].x = 400; pointArray[1].y = 0;

```

```

pointArray[2].x = 0; pointArray[2].y = 0;
pointArray[3].x = 0; pointArray[3].y = 400;

rc = MWXDb_Create_CellBoundary(cellId, numberOfPoint, pointArray);

if (rc == MWXSuccess) {
    MWX_Printf("New cellBoundary created\n");
} else {
    MWX_Printf("*** ERROR - Unable to create cellBoundary.\n");
    exit(MWXFail);
}
}

```

Scheme Equivalent

dbCreateCellBoundary (see Physical Implementation Online Help)

MWXDb_Create_CellInstance

Creates a cell instance in the cell identified by `cellId` at the position supplied by `placement`. The database identifier of the cell instance is returned in the `cellInstanceId`. All the transformations specified are applied to the new cell instance. The `cellInstanceId` function is undefined when the return code is not `MWXSuccess`.

Syntax

```

public int32 MWXDb_Create_CellInstance(
    MWXCellId_t cellId,           /* INPUT */
    char *childLibraryName,      /* INPUT */
    char *childCellName,         /* INPUT */
    char *cellInstanceName,      /* INPUT */
    char *rotationStr,           /* INPUT */
    char *mirrorStr,             /* INPUT */
    MWXPoint_t *placement,        /* INPUT */
    MWXObjId_t *cellInstanceId,   /* OUTPUT */
    int32 suppressMsg            /* INPUT */
)

```


Parameters

cellId	Specifies the database identifier of the cell in which to create the instance.
childLibraryName	Specifies the name of the library containing the master cell of the cell instance to create.
childCellName	Specifies the name of the master cell of the cell instance to create.
cellInstanceName	Specifies the instance name to apply to the new cell instance.
rotationStr	Specifies the counterclockwise rotation value in degrees (0, 90, 180, or 270).
mirrorStr	Specifies which direction in which to flip the cell instance (no, X, or Y).
placement	Specifies the x-y coordinate pair representing the location of the cell instance origin.
cellInstanceId	Specifies the database identifier of the new cell instance.
suppressMsg	Suppresses messages if TRUE.

Return

MWXSuccess	Occurs when the cell instance is successfully created.
MWXNoLibraryOpened	Occurs when a library is not open.
MWXFail	Occurs when the cell instance is not created for any other reason.

Example

```
char *libName = "design";
char* cellName = "top";
MWXCellId_t cellId;
MWXLibId_t libId;
char *childLibName = "refLib1";
char *childCellName = "cellA"
char *cellInstanceName = "cellA_1"
char *rotationStr = "0";
char *mirrorStr = "no";
MWXPoint_t origin;
MWXObjId_t cellInstId;
int32 suppressMsg = 1;
int rc;
```

```

origin.x = 200; origin.y = 0;
if (MWXDb_Open_Lib(libName, &libId) != MWXSuccess) {
    MWX_Printf("Fail to open library %s\n", libName);
    exit (MWXFail);
}
if (MWXDb_Open_Cell(cellName, 'w', &cellId) != MWXSuccess) {
    MWX_Printf("Fail to open cell %s\n", cellName);
    exit(MWXFail);
}
if (MWXDb_Create_CellInstance(cellId, childLibraryName, childCellName,
childInstanceName, rotationStr, mirrorStr, &origin, &cellInstanceId, TRUE) !=
MWXSuccess) {
    MWX_Printf("Fail to create cell Instance\n");
    exit (MWXFail);
}
MWX_Printf("Successfully create cellInstance\n");
/*
 * Close and save cell
 */

rc = MWXDb_Close_Cell_And_Save(cellId);
if (MWXSuccess == rc)
    MWX_Printf("Succeeded to close and save cell(%d)\n", cellId);
else
    MWX_Printf("Failed to close and save cell(%d) - rc is (%d)\n", cellId, rc);

/*
 * Close library
 */
rc = MWXDb_Close_Lib();
if (MWXSuccess == rc)
    MWX_Printf("Succeeded to close library\n");
else
    MWX_Printf("Failed to close library\n");

```

Scheme Equivalent

dbCreateCellLEQClass

(see Physical Implementation Online Help)

MWXDb_Create_CellInst_WithoutCheckStatus

Creates a cell instance object without checking the status.

Syntax

```

int MWXDb_Create_CellInst_WithoutCheckStatus(
MWXCellId_t cellId /* INPUT */
char *childLibraryName, /* INPUT */
char *childCellName, /* INPUT */
char *cellInstanceName, /* INPUT */
char *rotationStr, /* INPUT */
char *mirrorStr, /* INPUT */

```

```

MWXPoint_t *placement, /* INPUT */
MWXObjId_t *cellInstanceId /* OUTPUT */
)

```

Parameters

cellId	Specifies the cellId in the database
childLibraryName	Specifies the child library name
childCellName	Specifies the child cell Name
cellInstanceName	Specifies the name of the cellInst that you want to create
rotationStr	Specifies the rotation string
mirrorStr	Specifies the mirror string
placement	Specifies the coordinate of placement of cellInstance
cellInstanceId	Specifies the cellInstance db identifier number

Return

MWXSucccess	Occurs if successfully create cell instance.
MWXFail	Occurs if fail to create cell Instance

Example

```

MWXCellId_t cellId;
char *childLibraryName = "testLib;
char *childCellName = "testCell";
char *cellInstanceName = "inst1";
char *rotationStr = "0";
char *mirrorStr = "no";
MWXPoint_t *placement;

placement->x = 12;
placement->y = 6;
MWXObjId_t cellInstanceId;
int rc;
rc = MWXDb_Create_CellInst_WithoutCheckStatus(cellId, childLibraryName,
childCellName,
cellInstanceName, rotationStr, mirrorStr, &placement, &cellInstanceId);
switch (rc) {
    case MWXSucccess:
        MWX_Printf("Successfully create cellInstance without checking
        status\n");
        return (cellInstanceId);
    case MWXFail:
        printf("Fail to create cellInstance\n");

```

MWXDb_Create_CellInst_WithoutCheckStatus

```

        return (MWXFail);
    Default:
        printf("Unknown return code \n");
        return (MWXFail);
}

```

See Also

[MWXDb_Create_CellInstance](#)

MWXDb_Create_CellRegion

Creates a cell region and the region's database identifier.

Syntax

```

int MWXDb_Create_CellRegion (
    MWXCellId_t cellId, /* INPUT */
    MWXPoint_t pointArray[], /* INPUT: two points */
    MWXObjId_t *cellRegionId /* OUTPUT */
)

```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell.
<code>pointArray</code>	Two coordinates indicating the dimensions of the cell region.
<code>cellRegionId</code>	The database identifier of the created cell region.

Return

<code>MWXNoLibraryOpened</code>	Occurs if the library is not open.
<code>MWXFail</code>	Occurs if the cell region cannot be successfully created.
<code>MWXSuccess</code>	Occurs if the cell region is successfully created. The successful function should return the database identifier of the cell region.

Example

```

char *libName = "design";
char* cellName = "top";
MWXCellId_t cellId;
MWXLibId_t libId;
MWXPoint_t pointArray[4];
MWXObjId_t cellRegionId;
int rc;

```

```

pointArray[0].x = 0; pointArr[0].y = 0;
pointArray[1].x = 1000; pointArray[1].y = 0;
pointArray[2].x = 1000; pointArray[2].y = 2000;
pointArray[3].x = 0; pointArray[3].y = 2000;
if (MWXDb_Open_Lib(libName, &libId) != MWXSuccess) {
    MWX_Printf("Fail to open library %s\n",libName);
    exit (MWXFail);
}
if (MWXDb_Open_Cell(cellName,'w',&cellId) != MWXSuccess) {
    MWX_Printf("Fail to open cell %s\n",cellName);
    exit(MWXFail);
}
if (MWXDb_Create_CellRegion(cellId, pointArray, &cellInstanceId) !=
MWXSuccess) {
    MWX_Printf("Fail to create cell  region\n");
    exit (MWXFail);
}
MWX_Printf("Successfully create cellRegion\n");
/*
 * Close and save cell
 */
rc = MWXDb_Close_Cell_And_Save(cellId);
if (MWXSuccess == rc)
    MWX_Printf("Succeeded to close and save cell(%d)\n", cellId);
else
    MWX_Printf("Failed to close and save cell(%d) - rc is (%d)\n", cellId, rc);
/*
 * Close library
 */
rc = MWXDb_Close_Lib();
if (MWXSuccess == rc)
    MWX_Printf("Succeeded to close library\n");
else
    MWX_Printf("Failed to close library\n");

```

Scheme Equivalent

dbCreateCellRegion
(see Physical Implementation Online Help)

MWXDb_Create_CellRow

Creates a cell row.

Syntax

```

int MWXDb_Create_CellRow (
    MWXCellId_t cellId, /* INPUT */
    MWXPoint_t pointArray[], /* INPUT: two points */
    MWXObjId_t cellRegionId, /* INPUT */
    MWXObjId_t *cellRowId /* OUTPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell.
pointArray	Specifies the coordinates indicating the dimensions of the cell row.
cellRegionId	Specifies the database identifier of the cell region.
cellRowId	Returns the database identifier of the created cell row.

Return

MWXNoLibraryOpened	Occurs if the library is not open.
MWXFail	Occurs if the cell row cannot be successfully created.
MWXSuccess	Occurs if the cell row is successfully created. The successful function should return the database identifier of the cell row.

Example

```
char *libName = "design";
char* cellName = "top";
MWXCellId_t cellId;
MWXLibId_t libId;
MWXPoint_t pointArray[2];
MWXObjId_t cellRowId;
int rc;

pointArray[0].x = 0; pointArr[0].y = 0;
pointArray[1].x = 1000; pointArray[1].y = 1000;
    if (MWXDb_Open_Lib(libName, &libId) != MWXSuccess) {
        MWX_Printf("Fail to open library %s\n", libName);
        exit (MWXFail);
    }
if (MWXDb_Open_Cell(cellName, 'w', &cellId) != MWXSuccess) {
    MWX_Printf("Fail to open cell %s\n", cellName);
    exit(MWXFail);
}
if (MWXDb_Create_CellRow(cellId, pointArray, &cellInstanceId) != MWXSuccess) {
    MWX_Printf("Fail to create cell row\n");
    exit (MWXFail);
}
MWX_Printf("Successfully create cellRow\n");
/*
 * Close and save cell
 */

rc = MWXDb_Close_Cell_And_Save(cellId);
if (MWXSuccess == rc)
    MWX_Printf("Succeeded to close and save cell(%d)\n", cellId);
```

```

else
    MWX_Printf("Failed to close and save cell(%d) - rc is (%d)\n", cellId, rc);

/*
 * Close library
 */
rc = MWXDb_Close_Lib();
if (MWXSuccess == rc)
    MWX_Printf("Succeeded to close library\n");
else
    MWX_Printf("Failed to close library\n");

```

Scheme Equivalent

dbCreateCellRow

(see Physical Implementation Online Help)

MWXDb_Create_ContactCode

Creates a contact code from the given name and the technology information.

Syntax

```

int MWXDb_Create_ContactCode (
    MWXLibId_t libId, /* INPUT */
    char *contactCodeName, /* INPUT */
    MWXContactCodeTech_t contactCode, /* INPUT */
    MWXObjId_t contactCodeId /* OUTPUT */
)

```

Parameters

<code>libId</code>	Specifies the database identifier of the open library.
<code>contactCodeName</code>	Specifies the contact code name.
<code>contactCode</code>	Specifies the contact code technology information.
<code>contactCodeId</code>	Specifies the database identifier of the new contact code.

Return

<code>MWXSuccess</code>	Occurs if successfully creates the contact code.
<code>MWXFail</code>	Occurs if fails to create contact code.
<code>MWXNoLibraryOpened</code>	Occurs if there is no library currently open.

Example

```
MWXObjId_t create_contactCode (
    MWXLibId_t libId,
    char *contactCodeName,
    MWXContactCodeTech_t contactCode)
{
    MWXObjId_t contactCodeId;

    if (MWXDb_Create_ContactCode(libId, contactCodeName, contactCode,
    &contactCodeId) != MWXSuccess) {
        MWX_Printf("Fail to create contact Code\n");
        exit (MWXFail);
    }
    MWX_Printf("Successfully create contact Code\n");
    return (contactCodeId);
}
```

MWXDb_Create_ErrorComponents

Creates an error component (ErrorComponent) object and attaches it to an error object (ErrorObj). 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 an even-numbered list of coordinate pairs supplied to the function as the `pointArray` parameter. Each pair of coordinates represents opposing corners of a rectangle.

For example, for a point array containing four entries, the first two entries represent opposite corners of one rectangle, and the second two points represent opposite corners of a second rectangle.

Syntax

```
int MWXDb_Create_ErrorComponents (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t errorObjId, /* INPUT */
    char *netName, /* INPUT */
    int nodeNumber, /* INPUT */
    int16 numOfPoint, /* INPUT */
    MWXPoint_t pointArray[], /* INPUT */
    MWXObjId_t *errorComponentId /* OUTPUT */
)
```

Parameters

cellId	Represents the database identifier of the cell that owns the error object to which the error component is being attached.
errorObjId	Represents the database identifier of the error object to which the error component is being attached.
netName	Represents the name of the net to which the error component relates.
nodeNumber	Represents the integer value displayed by the verification tool (this value is not used by Synopsys tools).
numOfPoint	Represents the number of points in <code>pointArray</code> (must be an even number).
pointArray	Represents the array of points describing one or more rectangles (must be an even number of points, each pair representing opposite corners of a rectangle).
errorComponentId	Represents the database identifier of the new error component.

Return

MWXSuccess	Occurs if Milkyway successfully creates the error component.
MWXFail	Occurs if Milkyway fails to successfully create the error component.

Example

```
MWXObjId_t create_error_components (
    MWXCellId_t cellId,
    MWXObjId_t errorObjId,
    char *netName,
    int nodeNumber,
    int16 numOfPoint,
    MWXPoint_t pointArray[]
)
```

```

)
{
    MWXObjId_t errorComponentId;
    if (MWXDb_Create_ErrorComponents (cellId, errorObjId, netName, nodeNumber,
numOfPoint, pointArray, &errorComponentId) != MWXSuccess) {
        MWX_Printf("Fail to create error component object \n");
        return (MWXFail);
    }
    return (errorComponentId);
}

```

Scheme Equivalent

dbCreateErrorComponent

(see Physical Implementation Online Help)

MWXDb_Create_ErrorObj

Creates an error object (ErrorObj), which represents one error. The error object is associated with a single error type (ErrorType object) and might have numerous error components (ErrorComponent objects).

Syntax

```

int MWXDb_Create_ErrorObj (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t errorTypeId, /* INPUT */
    char *info, /* INPUT */
    MWXObjId_t *errorObjId /* OUT */
)

```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell that owns the error object.
<code>errorTypeId</code>	Specifies the database identifier of the type with which the error object is associated.
<code>info</code>	Specifies the information related to the error.
<code>errorObjId</code>	Specifies the database identifier of the new error object.

Return

<code>MWXSuccess</code>	Occurs if Milkyway successfully creates the error object.
<code>MWXFail</code>	Occurs if Milkyway fails to successfully create the error object.

Example

```
MWXObjID_t mwCreateErrorObj
(
    MWXCellId_t cellId,
    MWXObjId_t errorTypeId,
    char info[256]){
    MWXObjId_t errorObjId;
    int rc;

    MWX_Printf("INFO -- Create error object ...\n");

    rc = MWXDb_Create_ErrorObj (cellId, errorTypeId, info, &errorObjId);
    switch (rc) {
        case MWXSuccess:
            return (errorObjId);
        case MWXFail:
            MWX_Printf("ERROR -- Failed to create error object\n");
            return (MWXFail);
        default:
            MWX_Printf("ERROR -- unknown return code from MWXDb_Create_ErrorObj\n");
            return (MWXFail);
    }
}
```

Scheme Equivalent

`dbCreateGroup`

(see Physical Implementation Online Help)

MWXDb_Create_ErrorType

Creates an error type (ErrorType), which represents a particular kind of error and might have several related error objects.

Syntax

```
int MWXDb_Create_ErrorType (
    MWXCellId_t cellId, /* INPUT */
    char *summary, /* INPUT: max. 15 characters*/
    char *info, /* INPUT */
    unsigned char errorLevel, /* INPUT */
    MWXObjId_t *errorTypeId /* OUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that will own the error type.
summary	Specifies a brief description (15 characters or less) of the type.
info	Specifies information associated with the error type.
errorLevel	Specifies the severity of the error type (0 means informational, 1 means warning, 2 means error, 3 means severe error).
errorTypeId	Specifies the database identifier of the new error type.

Return

MWXSuccess	Occurs if Milkyway successfully creates the error type.
MWXFail	Occurs if Milkyway fails to successfully create the error type.

Example

```
MWXObjId_t mwCreateErrorType

MWXCellId_t cellId,
char summary[256],
char info[256],
unsigned char errorLevel){
MWXObjId_t errorTypeId;
int rc;

MWX_Printf("INFO -- cellId:      %d\n", cellId);
MWX_Printf("INFO -- summary:    %s\n", summary);
MWX_Printf("INFO -- info:         %s\n", info);
MWX_Printf("INFO -- errorLevel: %d\n", errorLevel);
```

```

MWX_Printf("INFO -- Creating an error type ...\n");

rc = MWXDb_Create_ErrorType (cellId, summary, info, errorLevel, &errorTypeId);
switch (rc) {
    case MWXSuccess:
        return (errorTypeId);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to create error type\n");
        return (MWXFail);
    default:
        MWX_Printf("ERROR -- unknown return code from MWXDb_Create_ErrorType\n");
        return (MWXFail);
}
}

```

Scheme Equivalent

dbCreateErrorType

(see Physical Implementation Online Help)

MWXDb_Create_Group

Creates a cell group identified by `groupName` and `groupSignature`.

Syntax

```

#include "mwx.h"
#include "mwx_proto.h"
int MWXDb_Create_Group (
    MWXCellId_t cellId, /* INPUT */
    int32 groupSignature, /* INPUT */
    char *groupName, /* INPUT */
    MWXObjId_t *groupId /* OUTPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell in which you want to create the group object.
groupSignature	Number you want to assign to the new group. Valid values: 0–255. A Synopsys application reserves group numbers 0–200, and 255 for predefined purposes.
groupName	Name you want to assign to the new cell group. Valid values: Any string with 31 characters or fewer.
groupId	Database identifier of the group you are creating.

Return

MWXFail	Occurs if the group is not successfully created for any reason, including out-of-range groupSignature.
MWXNoLicense	Occurs if the Synopsys license is not present.
MWXSuccess	Occurs if the group is successfully created.

Example

```
MWXObjId_t mwCreateGroup (
    MWXCellId_t cellId,
    int32 groupSignature,
    char *groupName
) {
    int rc;
    MWXObjId_t groupId;
    rc = MWXDb_Create_Group (cellId, groupSignature,
                             groupName, &groupId);

    switch (rc) {

        case MWXSuccess:
            return (groupId);
        case MWXFail:

            MWX_Printf("ERROR -- Failed to create group
                        '%s'\n", groupName);
            return (MWXFail);
        default:
            MWX_Printf("ERROR -- unknown return code from
                        MWXDb_Create_group\n");
            return (MWXFail);
    }
}
```

Scheme Equivalent

dbCreateGroup

(see Physical Implementation Online Help)

MWXDb_Create_Horizontal_Wire

Creates a horizontal wire in the cell identified by `cellId` at the specified location with the given length.

Syntax

```
int MWXDb_Create_Horizontal_Wire(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t wireMasterId, /* INPUT */  
    int32 wireLength, /* INPUT */  
    MWXPoint_t *point, /* INPUT */  
    MWXObjId_t *wireId /* OUTPUT */  
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell in which to create the horizontal wire.
<code>wireMasterId</code>	Specifies the database identifier of the wire master to use for the new wire.
<code>wireLength</code>	Specifies the length of the wire; a positive number extends the wire right of the origin, and a negative number extends the wire left of the origin.
<code>point</code>	Specifies the origin of the wire.
<code>wireId</code>	Specifies the database identifier of the new wire.

Return

<code>MWXSuccess</code>	Occurs if the wire is successfully created.
<code>MWXNoLibraryOpened</code>	Occurs if a library is not open.
<code>MWXFail</code>	Occurs if the wire is not successfully created for any other reason.

Example

```
MWXObjId_t mwCreateHorizontalWire (
```

```

    MWXCellId_t cellId,
    MWXObjId_t wireMasterId,
    int32 wireLength,
    MWXPoint_t point[] )
{
    int rc;
    MWXObjId_t hwireId;
    rc = MWXDb_Create_Horizontal_Wire (cellId, wireMasterId, wireLength, &point,
    &wireId);
    switch (rc) {
        case MWXSuccess:
            return (hwireId);
        case MWXFail :
            MWX_Printf("Fail to create horizontal wire\n");
            return (MWXFail);
        default:
            return (MWXFail);
    }
}

```

Scheme Equivalent

dbCreateHorizontalWire
(see Physical Implementation Online Help)

MWXDb_Create_Lib

Creates a new library with the given library name using the `techFileName` technology file with the given hierarchy separator.

Syntax

```

int MWXDb_Create_Lib(
    char *libraryName, /* INPUT */
    char *techFileName, /* INPUT */
    char *hierSeparator /* INPUT */
)

```


Parameters

libraryName	Specifies the desired name of the library.
techFileName	Specifies the name of the technology file used to create the library.
hierSeparator	Specifies the hierarchy separator.

Return

MWXSucces	Occurs if the library is successfully created.
MWXLibraryAlready Opened	Occurs if a library is already open.
MWXFail	Occurs if the library is not successfully created.

Description

The `MWXDb_Create_Lib` function creates a new library with the given library name using `techFileName` with the given hierarchy separator. The first character of `hierSeparator` is used as the hierarchy separator. If an empty string or NULL is passed in for `hierSeparator`, the default separator "/" is used. If the library name already exists, then `MWXFail` is returned. On success, the library opens in write mode.

Example

```
int CreateLib(char *libraryName, char *techFileName, char
*hierSeparator) {

    int rc;

    MWX_Printf("INFO -- MWXDb_Create_Lib.\n");
    rc = MWXDb_Create_Lib(libraryName, techFileName,
                          hierSeparator);

    switch (rc) {
    case MWXSucces:
        MWX_Printf("Successfully created library. Library is
now open in write mode.\n");
        return (MWXSucces);
    case MWXLibraryAlreadyOpened:
        MWX_Printf("ERROR -- Library already opened. Please
close the current design library first. MWXDb_Create_Lib
failed.\n");
        return (MWXFail);
    case MWXFail:
        MWX_Printf("ERROR -- Unable to create library.
MWXDb_Create_Lib failed.\n");
        return (MWXFail);
    default:
```

```

        MWX_Printf("UNKNOWN ERROR %d from MWXDb_Create_Lib.\n",
                    rc);
        return (MWXFail);
    }
}

```

Scheme Equivalent

dbCreateLib

(see Physical Implementation Online Help)

See Also

[MWXDb_Rename_Lib](#)

[MWXDb_Copy_Lib](#)

[MWXDb_Get_CurrentLibId](#)

[MWXDb_Open_Lib](#)

[MWXDb_Open_Library_ByAccessMode](#)

[MWXDb_Close_Lib](#)

MWXDb_Create_LibAttachFile

Creates a UNIX file attached to an opened library. After creation, the file is open for write operation. You are responsible for maintaining the attached file after the creation.

Syntax

```

FILE *MWXDb_Create_LibAttachFile (
    char *pseudoAccessName /* INPUT */
)

```

Parameters

pseudoAccessName	Identifies the attached file for future access. It must have a unique name in a library.

Return

FILE *	The file pointer is returned, if the file is successfully created.
MWXFail	Occurs if file cannot be successfully created.

Example

```

fp = MWXDb_Create_LibAttachFile ( fileName );
if( fp != NULL )
{

```

```

MWX_Printf("INFO -- File '%s' created\n",fileName);
}
else
{
MWX_Printf("ERROR -- Failed to create Lib attached file\n");
return (MWXFail);
}

```

MWXDb_Create_Net

Creates a net with the name given by `netName` in the cell identified by `cellId`. The database identifier of the net is returned in `netId`. Note that `netId` is undefined if the return code is not `MWXSuccess`.

Syntax

```

int MWXDb_Create_Net(
    MWXCellId_t cellId, /* INPUT */
    char *netName, /* INPUT */
    MWXObjId_t *netId /* OUTPUT */
)

```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell in which to create the net.
<code>netName</code>	Specifies the name of the net to create.
<code>netId</code>	Specifies the database identifier of the new net.

Return

<code>MWXSuccess</code>	Occurs if the net is successfully created.
<code>MWXFail</code>	Occurs if the net is not successfully created.
<code>MWXNoLibraryOpened</code>	Occurs if a library is not currently open.

Example

```

MWXObjId_t mwCreateNet (
    MWXCellId_t cellId,
    char *netName )
{
    int rc;
    MWXObjId_t netId;
    rc = MWXDb_Create_Net (cellId, netName, &netId);
    switch (rc) {
        case MWXSuccess:

```

```

        MWX_Printf("Successfully create net %s\n",netName);
        return (netId);
    case MWXFail:
        MWX_Printf("ERROR -- failed to create net\n");
        return (MWXFail);
    case MWXNoLibraryOpened:
        MWX_Printf("ERROR -- library not open\n");
        return (MWXFail);
    }}

```

Scheme Equivalent

dbCreateNet

(see Physical Implementation Online Help)

MWXDb_Create_ParameterSet

Creates a parameter set record in a cell.

Syntax

```

int MWXDb_Create_ParameterSet (
    MWXCellId_t cellId, /* INPUT */
    char *appName, /* INPUT */
    MWXObjId_t *objectId /* OUTPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell to which information applies.
appName	The name of the parameter set.
objectId	Specifies the database identifier of the parameter set.

Return

MWXFail	Occurs if parameterSet is not successfully created.
MWXSuccess	Occurs if the parameterSet is successfully created.

Example

```

MWXObjId_t mwCreateParameterSet (
    MWXCellId_t cellId,
    char *paraName
) {
    int rc;
    MWXObjId_t parameterSetId;

```

```

rc = MWXDb_Create_ParameterSet ( cellId, appName, &objectId );
    switch (rc) {
        case MWXSuccess:
            MWX_Printf("Successfully create parameterSet
                        %s\n",paraName);
            return (parameterSetId);
        case MWXFail:
            MWX_Printf("ERROR -- Failed to operate. \n");
            return (MWXFail);
    }

```

Scheme Equivalent

dbCreateParamSet

(see Physical Implementation Online Help)

MWXDb_Create_Path

Creates a path with the given layerName, dataType, pathType, routeType, width, numberOfPoints, and pointArray in the specified cell. The database identifier of the new path is returned in pathId.

Syntax

```

int MWXDb_Create_Path(
    MWXCellId_t cellId, /* INPUT */
    char *layerName, /* INPUT */
    uchar dataType, /* INPUT */
    uchar pathType, /* INPUT */
    uchar routeType, /* INPUT */
    uint32 width, /* INPUT */
    uint32 numberOfPoints, /* INPUT */
    MWXPoint_t pointArray[], /* INPUT */
    MWXObjId_t *pathId /* OUTPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell in which to create the path.
layerName	Specifies the layer name on which to create the path.
dataType	Specifies the data type of the created path. Same as GDSII dataType; the default value is 0.
pathType	Specifies the type of the created path. If 0, the path edge extends to the coordinate. If 2, the path edge extends a half width beyond the coordinate.
routeType	Specifies the route type of the created path.
width	Specifies the desired width for the created path.
numberOfPoints	Specifies the number of points used to define the path.
pointArray	Specifies the array of x/y-coordinate pairs of points.
pathId	Returns the object identifier of the new path.

Return

MWXSuccess	Occurs if the path is successfully created.
MWXNoLibraryOpened	Occurs if the design library is not open.
MWXFail	Occurs if the path is not successfully created for any other reason.

Dynamic Example

```
MWXSCM_t mwCreatePath(MWXSCM_t list) {  
  
    MWXCellId_t cellId;  
    int32 rc;  
    MWXSCM_t rtn;  
    MWXObjId_t pathId;  
    char layerName[256];  
    int32 dataType, pathType, routeType;  
    uint32 width, numOfPoints;  
    MWXPoint_t *pointArray = NULL;  
    MWXSCM_t pointList;  
    MWXSCM_t point;  
    MWXSCM_t l;  
    double x, y;  
  
    if (MWXSc_Scanf(list, "osddddl", &cellId, layerName, &dataType, &pathType,
```

```

        &routeType, &width, &pointList) != 7) {
MWX_Printf("ERROR -- mwCreatePath requires 7 arguments.\n");
return (MWX_FALSE_SCM);
}

    if (! MWXSc_Is_ListSCM(pointList)) {
MWX_Printf("ERROR -- improper point list.\n");
return (MWX_FALSE_SCM);
}

numOfPoints = 0;
for (l = pointList; ! MWXSc_Is_NullSCM(l); l = MWXSc_Cdr(l)) {

    /* Get the next coordinate pair. Make sure its a list. */
    point = MWXSc_Car(l);
    if (! MWXSc_Is_ListSCM(point)) {
        MWX_Printf("ERROR -- improper point.\n");
        free (pointArray);
        return (MWX_FALSE_SCM);
    }

    /* Get the two coordinates. */
    if ((MWXSc_Get_DoubleSCM(MWXSc_Car(point), &x) == MWXFail) ||
        (MWXSc_Get_DoubleSCM(MWXSc_Car(MWXSc_Cdr(point)), &y) == MWXFail)) {
        MWX_Printf("ERROR -- bad point found at position %d.\n", numOfPoints);
        free (pointArray);
        return (MWX_FALSE_SCM);
    }

    /* Add the coordinates to the point array. */
    numOfPoints++;
    pointArray = (MWXPoint_t*) realloc (pointArray,
        sizeof(MWXPoint_t) * numOfPoints);
    pointArray[numOfPoints-1].x = (int32) x;
    pointArray[numOfPoints-1].y = (int32) y;
}

MWX_Printf("INFO -- MWXDb_Create_Path.\n");
rc = MWXDb_Create_Path (cellId, layerName, dataType, pathType, routeType,
    width, numOfPoints, pointArray, &pathId);

switch (rc) {
case MWXSuccess:
    MWX_Printf("Successfully created path. Path Id = %d.\n", pathId);
    rtn = MWXSc_Make_DBIIdSCM (pathId);
    break;
case MWXNoLibraryOpened:
    MWX_Printf("ERROR -- No library opened. MWXDb_Create_Path failed.\n");
    rtn = MWX_FALSE_SCM;
    break;
case MWXFail:
    MWX_Printf("ERROR -- Unable to create path. MWXDb_Create_Path failed.\n");
    rtn = MWX_FALSE_SCM;
    break;
default:
    MWX_Printf("UNKNOWN ERROR %d from MWXDb_Create_Path.\n", rc);
    rtn = MWX_FALSE_SCM;
}

```

```

}

free (pointArray);
return (rtn);
}

Example

int CreatePath(MWXCellId_t cellId, char* layerName, int32 dataType, int32
               pathType, int32 routeType, uint32 width, uint32 numberOfPoints,
               MWXPoint_t pointArray[], MWXObjId_t *pathId) {
int rc;

MWX_Printf("INFO -- MWXDb_Create_Path.\n");
rc = MWXDb_Create_Path(cellId, layerName, dataType, pathType, routeType, width,
                       numberOfPoints, pointArray, pathId);

switch (rc) {
    case MWXSuccess:
        MWX_Printf("Successfully created path. Path ID = %d.\n", *pathId);
        return (MWXSuccess);
    case MWXNoLibraryOpened:
        MWX_Printf("ERROR -- No library opened. MWXDb_Create_Path
                   failed.\n");
        return (MWXFail);
    case MWXFail:
        MWX_Printf("ERROR -- Unable to create path. MWXDb_Create_Path
                   failed.\n");
        return (MWXFail);
default:
        MWX_Printf("UNKNOWN ERROR %d from MWXDb_Create_Path.\n", rc);
        return (MWXFail);
}
}

```

Scheme Equivalent

dbCreatePath
(see Physical Implementation Online Help)

See Also

[Path Object and Field Descriptions](#)
[MWXDb_Convert_PathToPolygon](#)
[MWXDb_Convert_PathToRectangles](#)
[MWXDb_Convert_ObjectsToPath](#)

MWXDb_Create_Pin

Creates a pin with the name given by `pinName` in the cell identified by `cellId` and attached to the net identified by `netId`. The database identifier of the pin is returned in `pinId`. Note that `pinId` is undefined if the return code is not `MWXSuccess`.

Syntax

```
int MWXDb_Create_Pin(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t netId, /* INPUT */  
    char *pinName, /* INPUT */  
    char *layerName, /* INPUT */  
    int32 numberOfPoints, /* INPUT */  
    MWXPoint_t pointArray[], /* INPUT */  
    MWXObjId_t *pinId /* OUTPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell in which to create the pin.
netId	Specifies the database identifier of the net to which the pin is attached.
pinName	Specifies the name of the new pin.
layerName	Specifies the name of the layer from the technology file on which the pin resides.
numberOfPoints	Specifies the number of x/y-coordinate pairs contained in pointArray.
pointArray	Specifies the set of x/y-coordinate pairs defining the pin boundary.
pinId	Specifies the database identifier of the new pin.

Return

MWXSuccess	Occurs if the pin is successfully created.
MWXNoLibraryOpened	Occurs if a library is not open.
MWXFail	Occurs if the pin is not successfully created for any other reason.

Example

```
MWXObjId_t mwCreatePin (  
    MWXCellId_t cellId,  
    MWXObjId_t netId (  
{  
    char *pinName = "IN";  
    char *layerName = "M2";  
    int rc;  
    int32 numberOfPoints = 2;
```

```

MWXObjId_t pinId;
MWXPoint_t pointArray[2];

pointArray[0].x = 1;
pointArray[0].y = 2;
pointArray[1].x = 5;
pointArray[1].y = 5;
rc = MWXDb_Create_Pin (cellId, netId, pinName,
                      layerName, numberOfPoints, pointArray, &pinId);
switch (rc) {
    case MWXSuccess:
        MWX_Printf("Successfully create pin with pinName = %s\n",pinName);
        return (pinId);
    case MWXFail:
        MWX_Printf("ERROR -- failed to create pin\n");
        return (MWXFail);
    case MWXNoLibraryOpened:
        MWX_Printf("ERROR -- library not open\n");
        return (MWXFail);
}}

```

Scheme Equivalent

dbCreatePin

(see Physical Implementation Online Help)

MWXDb_Create_PinWithPortName

Create a pin with the given `portName` in the cell identified with a `cellId` and attached to the net identified by a `netId`.

Syntax

```

public int32
MWXDb_Create_PinWithPortName(
    MWXCellId_t cellId, /* IN: identify number of cell */
    MWXObjId_t netId, /* IN: net db identifier number */
    char *portName, /* IN: port name that are used to
                     create pin */
    char *pinName, /* IN: pin name */
    char *layerName, /* IN: layer name that the path
                     belongs to unit32 */
    numberOfPoint, /* IN: number of points that are used
                     to create pin */
    MWXPoint_t pointArray[], /* IN: coordinate of points that
are used to create pin */
    MWXObjId_t *pinId /* OUT: pin db identifier number. */
)

```

Parameters

cellId	Specifies the database identifier of the cell in which to create the pin.
netId	Specifies the database identifier of the net to which the pin is attached.
portName	Specifies the name of the given port to which the pin will be connected.
pinName	Specifies the name of the new pin.
layerName	Specifies the name of the layer from the technology file on which the pin resides.
numberOfPoint	Specifies the number of x/y-coordinate pairs contained in pointArray.
pointArray	Specifies the set of x/y-coordinate pairs defining the pin boundary.
pinId	Specifies the database identifier of the new pin.

Return

MWXSuccess	Occurs when the command successfully creates a pin with a port.
MWXFail	Occurs when the command fails to successfully create a pin with a port.

Example

```
MWXObjId_t mwCreatePin (
    MWXCellId_t cellId,
    MWXObjId_t netId (
{
    char *pinName = "IN";
    char *layerName = "M2";
    int rc;
    int32 numberOfPoints = 2;
    MWXObjId_t pinId;
    MWXPoint_t pointArray[2];

    pointArray[0].x = 1;
    pointArray[0].y = 2;
    pointArray[1].x = 5;
    pointArray[1].y = 5;
    MWXCellId_t cellId;
    MWXObjId_t netId;
    char *cellName = "testCell";
```

```

        char *netName = "IN";
        char *portName = "ir-port-n(2)";
        char *pinName = "ir(2)";
        char *layerName="M2R";
        int32 numberOfPoint = 2;
        MWXPoint_t *pointArray;
        MWXObjId_t pinId;
        int rc;

if (MWXDb_Open_Cell (cellName, "w", &cellId) != MWXSuccess) {
    MWX_Printf("Fail to open the cell for write\n");
    exit (MWXFail);
if (MWXDb_Get_Net_ByName (cellId, netName, &netId) != MWXSuccess) {
    MWX_Printf("Fail to get net by name \n");
    exit (MWXFail);
}

pointArray[0].x = 0.0;
pointArray[0].y = 271.5;
pointArray[1].x = 3.4;
pointArray[1].y = 272.5;

rc = MWXDb_Create_PinWithPortName (cellId, netId, portName,pinName,
                                   layerName,numberOfPoint, pointArray, &pinId);
switch (rc) {
    case MWXSuccess:
        MWX_Printf("Successfully create pin with portName
                  %s\n",portName);
        return (pinId);
    case MWXFail:
        MWX_Printf("Fail to create pin with portName %s\n",portName);
        return (MWXFail);
    default:
        return (MWXFail);
}

free (pointArray);

```

MWXDb_Create_PlanGroup

Creates a PlanGroup object.

Syntax

```

int MWXDb_Create_PlanGroup (
    MWXCellId_t cellId, /* INPUT */
    char *groupName, /* INPUT */
    int32 groupType, /* INPUT */
    int32 targetPercent, /* INPUT */
    int32 lowLeftX, /* INPUT */
    int32 lowLeftY, /* INPUT */
    int32 upRightX, /* INPUT */
    int32 upRightY, /* INPUT */
    MWXObjId_t *groupId /* OUTPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell that owns the PlanGroup object.
groupName	Name of the PlanGroup object.
groupType	Specifies an integer that represents a routetype. The valid routetype is 1.
targetPercent	Specifies the area utilization percentage.
lowLeftX	Specifies the lower left x-coordinate of the planGroup.
lowLeftY	Specifies the lower left y-coordinate of the planGroup.
upRightX	Specifies the upper right x-coordinate of the planGroup.
upRightY	Specifies the upper right y-coordinate of the planGroup.
groupId	Specifies the database identifier of the PlanGroup object.

Return

MWXNoLicense	Occurs if no software license is available.
MWXSuccess	Occurs if the PlanGroup object is successfully created.
MWXFail	Occurs if the PlanGroup object is not successfully created.

Example

```
char *cellName = "testCell";
MWXCellId_t cellId;
char *groupName = "groupName";
int32 groupType = 1;
int32 targetPercent = 50;
int32 lowLeftX = 6;
int32 lowLeftY = 12;
int32 upRightX = 12;
int32 upRightY = 12;

int rc;

if (MWXDb_Open_Cell(cellName,"w", &cellId) != MWXSuccess) {
    MWX_Printf("Fail to open cell %s\n",cellName);
    exit(MWXFail);
}
rc = MWXDb_Create_PlanGroup( cellId, groupName, groupType, targetPecent,
    lowLeftX, lowLeftY, upRightX, upRightY &groupId );
```

```

switch (rc) {
    case MWXSuccess:
        MWX_Printf("Successfully create planGroup\n");
        return (groupId);
    case MWXFail:
        MWX_Printf("Fail to create planGroup\n");
        return (MWXFail);
}

```

MWXDb_Create_Polygon

Creates a polygon in the cell identified by `cellId` on the layer given by `layerName` with the desired number of points from `pointArray` given by `numberOfPoints`. The database identifier of the new polygon is returned in `polygonId`.

Syntax

```

int MWXDb_Create_Polygon(
    MWXCellId_t cellId, /* INPUT */
    char *layerName, /* INPUT */
    int32 numberOfPoints, /* INPUT */
    MWXPoint_t pointArray[], /* INPUT */
    MWXObjId_t *polygonId /* OUTPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell from which to create the polygon.
layerName	Specifies the name of the layer from the technology file on which the polygon resides.
numberOfPoints	Specifies the number of x/y-coordinate pairs contained in pointArray.
pointArray	Specifies the set of x/y-coordinate pairs defining the polygon boundary.
polygonId	Returns the database identifier of the new polygon.

Return

MWXSuccess	Occurs if the polygon is successfully created.
MWXNoLibraryOpened	Occurs if the design library is not open.
MWXFail	Occurs if the polygon is not successfully created for any other reason.

Example

```
int CreatePolygon(MWXCellId_t cellId, char *layerName, int32 numberOfPoints,
    MWXPoint_t pointArray[], MWXObjId_t *polygonId) {

    int rc;

    MWX_Printf("INFO -- MWXDb_Create_Polygon.\n");
    rc = MWXDb_Create_Polygon (cellId, layerName, numberOfPoints, pointArray,
        polygonId);

    switch (rc) {
    case MWXSuccess:
        MWX_Printf("Successfully created polygon.  Polygon Id = %d.\n",
            *polygonId);
        return (MWXSuccess);
    case MWXNoLibraryOpened:
        MWX_Printf("ERROR -- No library opened.  MWXDb_Create_Polygon failed.\n");
        return (MWXFail);
    case MWXFail:
        MWX_Printf("ERROR -- Unable to create Polygon.  MWXDb_Create_Polygon
            failed.\n");
        return (MWXFail);
    default:
        MWX_Printf("UNKNOWN ERROR %d from MWXDb_Create_Polygon.\n", rc);
        return (MWXFail);
    }
}
```

Scheme Equivalent

`dbCreatePolygon`

(see Physical Implementation Online Help)

See Also

[Polygon Object](#) and Field Descriptions

[MWXDb_Convert_PathToPolygon](#)

[MWXDb_Convert_PolygonToRectangles](#)

[MWXDb_Convert_RectangleToPolygon](#)

[MWXDb_Convert_ObjectsToPath](#)

MWXDb_Create_Port

Creates a port with the given direction for the net of the specified cell. Basic port directions defined in the `mwX.h` header file include `MWXCUnknownDirectionPort`, `MWXCInputPort`, `MWXCOutputPort`, `MWXCInputOutputPort`, and `MWXCTriState`.

Syntax

```
int MWXDb_Create_Port(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t netId, /* INPUT */  
    char *portName, /* INPUT */  
    uchar direction, /* INPUT */  
    MWXObjId_t *portId /* OUTPUT */  
)
```


Parameters

cellId	Specifies the database ID of the cell in which to create the port.
netId	Specifies the database ID of the net in which to create the port.
portName	Specifies the name of the port you are creating.
direction	Specifies the direction of the port you are creating.
portId	Returns the database ID of the port created.

Return

MWXSuccess	Occurs if the port is created successfully.
MWXNoLibraryOpened	Occurs if port fails to be created because the design library is not open.
MWXFail	Occurs if port fails to be created for any other reason.

Example

```
/* =====
 *   Creating an Input/Output port
 *   ===== */

int CreatePort(MWXCellId_t cellId, MWXObjId_t netId, char *portName, MWXObjId_t
               *portId) {
    int rc;
    uchar direction = MWXcInputOutputPort;

    MWX_Printf("INFO -- MWXDb_Create_Port.\n");
    rc = MWXDb_Create_Port (cellId, netId, portName, direction, portId);

    switch (rc) {
    case MWXSuccess:
        MWX_Printf("Successfully created port, port id = %d\n", *portId);
        return (MWXSuccess);
    case MWXNoLibraryOpened:
        MWX_Printf("ERROR -- No design library opened.\n");
        return (MWXFail);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to create port\n");
        return (MWXFail);
    default:
        MWX_Printf("ERROR -- unknown return code from MWXDb_Create_Port\n");
        return (MWXFail);
    }
}
```

Scheme Equivalent

dbCreatePort

MWXDb_Create_Property

Creates a property with the name given by `propertyName` and assigns the value given by `propertyValue`. Attaches the property to the object identified by `objId`. The `propertyId` is undefined if the return code is not `MWXSuccess`.

Syntax

```
int MWXDb_Create_Property(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t objectId, /* INPUT */  
    char *propertyName, /* INPUT */  
    unsigned char propertyDataType, /* INPUT */  
    MWXPropValue_t *propertyValue, /* INPUT */  
    uint16 propertySize, /* INPUT */  
    MWXObjId_t *propertyId /* OUTPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell containing the object for which to create the property.
objectId	Specifies the database identifier of the object for which to create the property.
propertyName	Specifies the name of the property to create.
propertyDataType	Specifies the type of the property to create; valid values can be found in the mwx.h file: MWXcPropBooleanTypeMWXcPropIntegerTypeMWXcPropFloatingTypeMWXcPropStringTypeMWXcPropDataStructTypeMWXcPropTimeType
propertyValue	Specifies the value to assign to the new property.
propertySize	Specifies the memory size, in bytes, of the property value (for strings, use the string length; for all others, use the size of the data type).
propertyId	Specifies the database identifier of the new property.

Return

MWXSuccess	Occurs if Milkyway successfully creates the property.
MWXFail	Occurs if Milkyway fails to successfully create the property.

Example

```
MWXObjId_t propId;
MWXPoint_t pointArray[4];
MWXPropValue_t propertyValueNet;
MWXObjId_t pinId;

/** Create cell Prop */
NET myNet = {"IN",          /* net name */
            1245 /* netId */
            };

propertyValueNet.pStruct = &myNet;

/** Create Pin object */
netId = 1245;
layerName = "M2R";
pointArray[0].x = 0; pointArray[0].y = 0;
pointArray[1].x = 100; pointArray[1].y = 0;
pointArray[2].x = 100; pointArray[2].y = 100;
pointArray[3].x = 0; pointArray[3].y = 100;
```

```

rc = MWXDb_Create_Pin(cellId, netId, "IN1", layerName, 4, pointArray, &pinId);
if (rc != MWXSuccess) {
    MWX_Printf("Fail to create pin\n");
    return (MWXFail);
}

rc = MWXDb_Create_Property(cellId, pinId, "original Net",
MWXcPropDataStructType,
    &propertyValueNet, sizeof(NET), &propId);
if (rc != MWXSuccess) {
    exit(MWXFail);
}
return (MWXSuccess);

```

Scheme Equivalent

dbCreateProperty

(see Physical Implementation Online Help)

MWXDb_Create_Rectangle

Creates a rectangle in the cell identified by `cellId` on the desired layer given by `layerName`. The lower-left and upper-right corners of the rectangle are specified by the first two points in `pointArray` respectively; `pointArray` must contain exactly two points. The database identifier of the rectangle is returned in `objectId`. Note that `objectId` is undefined if the return code is not `MWXSuccess`.

Syntax

```

int MWXDb_Create_Rectangle(
    MWXCellId_t cellId, /* INPUT */
    char *layerName, /* INPUT */
    MWXPoint_t pointArray[], /* INPUT */
    MWXObjId_t *objectId /* OUTPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell in which to create the rectangle.
layerName	Specifies the name of the layer on which the rectangle will reside.
pointArray	Specifies the pair of x/y-coordinate pairs representing the lower-left and upper-right corners of the rectangle.
objectId	Returns the database identifier of the new rectangle.

Return

MWXSucces	Occurs if the rectangle is successfully created.
MWXNoLibraryOpened	Occurs if the design library is not open.
MWXFail	Occurs if the rectangle is not successfully created for any other reason.

Example

```
int CreateRectangle(MWXCellId_t cellId, char *layerName, MWXPoint_t
                    pointArray[], MWXObjId_t *objectId) {

int rc;

MWX_Printf("INFO -- MWXDb_Create_Rectangle.\n");
rc = MWXDb_Create_Rectangle (cellId, layerName, pointArray, objectId);

switch (rc) {
case MWXSucces:
MWX_Printf("Successfully created rectangle.  Rectangle Id = %d.\n",
            *objectId);
return (MWXSucces);
case MWXNoLibraryOpened:
MWX_Printf("ERROR -- No library opened.  MWXDb_Create_Rectangle
            failed.\n");
return (MWXFail);
case MWXFail:
MWX_Printf("ERROR -- Unable to create rectangle.  MWXDb_Create_Rectangle
            failed.\n");
return (MWXFail);
default:
MWX_Printf("UNKNOWN ERROR %d from MWXDb_Create_Rectangle.\n", rc);
return (MWXFail);
}
}
```

Scheme Equivalent

dbCreateRectangle

(see Physical Implementation Online Help)

See Also

[Rectangle Object](#) and Field Descriptions

[MWXDb_Convert_PathToRectangles](#)

[MWXDb_Convert_PolygonToRectangles](#)

[MWXDb_Convert_RectangleToPolygon](#)

[MWXDb_Convert_ObjectsToPath](#)

MWXDb_Create_TdfPad

Allows setting of pad placement constraints in the design.

Syntax

```
int MWXDb_Create_TdfPad (
    MWXCellId_t cellId, /* INPUT */
    char *padName, /* INPUT */
    char *padSide, /* INPUT */
    int32 padOrder, /* INPUT */
    int32 padOffset, /* INPUT */
    char *reflect /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that owns the pads.
padName	Name of the pad to be constrained (can be hierarchical).
padSide	Specifies the side or the corner of the chip on which the pad should be placed. Valid values are right, left, top, bottom, and none. See the <i>Milkyway Environment Data Preparation User Guide</i> for more details.
padOrder	Pad placement order relative to that of the other pads on the same side. Valid values: 0–32767. See the <i>Milkyway Environment Data Preparation User Guide</i> for more details.
padOffset	Pad placement relative offset from the bottom edge or left edge of the cell boundary. Valid values: Any number (in distance units) specifying the distance within the cell boundary that includes the pad width, or a negative number to indicate no offset constraint on the pad. See <i>Milkyway Environment Data Preparation User Guide</i> for more details.
reflect	The valid value is the keyword <code>reflect</code> , to indicating that the pad should be flipped. If pad flipping is not desired, omit this parameter.

Return

MWXNoLicense	Occurs if no software license is available.
MWXSuccess	Occurs if the create pad operation is successful.
MWXFail	Occurs if the create pad operation is unsuccessful.

Example

```
int32 mwCreateTdfPad (
    char      *libName,
    char      *cellName,
    char      *tdfFileName )
{
    MWXLibId_t  libId;
    MWXCellId_t cellId;
    char        ref[30];
    char        c, str1[30], str2[30], str3[30], str4[30];
    char        padName[30], padSide[30], reflect[30];
    int         padOrder, padOffset, i, j;
    FILE        *fPtr;

    if ((rc = MWXDb_Open_Lib (libName, &libId)) != MWXSuccess) {
        if (rc == MWXLibraryAlreadyOpened)
            MWX_Printf(" ***** library %s already open ***** \n", libName);
        else {
            MWX_Printf(" ***** library open %s failed ***** \n", libName);
            exit(MWXFail);
        }
    }
```

MWXDb_Create_TdfPad

```

    }
}
if ((fPtr = fopen(fileName,"r")) == NULL) {
    MWX_Printf("*****File could not be opened*****\n");
    exit (MWXFail);
}
else {
    MWX_Printf(" ***** Retrieving input data from tdf file*****\n");
    c = fgetc(fPtr);
    while (!feof(fPtr) && (c == ';'))
    {
        ungetc(c,fPtr);
        fgets(str1,80,fPtr);
        c = fgetc(fPtr);
    }
    ungetc(c,fPtr);

    rc = MWXDb_Open_Cell (cellName, 'w', &cellId);

    if ((rc != MWXSuccess) && (rc != MWXCellAlreadyOpened)) {
        MWX_Printf(" ***** cell open %s failed ***** \n", cellName);
        MWXDb_Close_Lib();
        exit (MWXFail);
    }
    if (rc == MWXCellAlreadyOpened)
        MWX_Printf(" ***** cell %s already open ***** \n", cellName);

    while (!feof(fPtr)) {
        str4[0] = '\0';
        padOffset = 0;
        fscanf(fPtr,"%s%s%s%d%s", str1, str2,str3,&padOrder,&padOffset,str4);
        if (padOffset > 32767)
            padOffset = 0;
        for (i=0,j=0;str2[i] != '\0';i++) {
            if (str2[i] != '') {
                padName[j] = str2[i];
                j++;
            }
        }
        padName[j] = '\0';
        for (i=0,j=0;str3[i] != '\0';i++) {
            if (str3[i] != '') {
                padSide[j] = str3[i];
                j++;
            }
        }
        padSide[j] = '\0';

        if (str4[0] == 0)
            reflect[0] = '\0';
        else {
            if (strcmp(str4,"\"reflect\"") == 0)
                strcpy(reflect,"reflect");
            else if (strcmp(str4, "\"optimizeReflect\"") == 0)
                strcpy(reflect,"optimizeReflect");
            else
                reflect[0] = '\0';
        }
        if (MWXDb_Create_TdfPad (cellId,

```



```

padName,padSide,padOrder,padOffset,reflect) != MW
XSuccess) {
    MWX_Printf(" ***** create pad failed ***** \n");
    MWXDb_Close_Lib();
    return (MWXFail);
}

}

if ( MWXDb_Close_Cell(cellId) != MWXSuccess) {
    MWX_Printf("Can't save cell\n");
    MWXDb_Close_Lib();
    exit(MWXFail);
}
MWXDb_Close_Lib();
return (MWXSuccess);
}
}

```

MWXDb_Create_Text

Creates a text object in the cell identified by `cellId` at the position supplied by `placement`. The database identifier of the new text object is returned in `textId`. All the transformations specified are applied to the new text object. The `textId` is undefined if the return code is not `MWXSuccess`.

Syntax

```

int MWXDb_Create_Text (
    MWXCellId_t cellId, /* INPUT */
    char *layerName, /* INPUT */
    char *justStr, /* INPUT */
    char *rotationStr, /* INPUT */
    char *mirrorStr, /* INPUT */
    int32 height, /* INPUT: in user units */
    MWXPoint_t *placement, /* INPUT: in user units */
    char *textStr, /* INPUT */
    MWXObjId_t *textId /* OUTPUT */
)

```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell in which to create text.
<code>layerName</code>	Specifies the name of the layer on which the new text will reside.
<code>justStr</code>	Defines placement of text around origin with legal values as follows: <code>ll</code> - places text at the lower left of the origin <code>lc</code> - places text at the lower center of the origin <code>lr</code> - places text at the lower right of the origin <code>cl</code> - places text at the center left of the origin <code>cc</code> - centers text around the origin <code>cr</code> - places text at the center right of the origin <code>ul</code> - places text at the upper left of the origin <code>uc</code> - places text at the upper center of the origin <code>ur</code> - places text at the upper right of the origin
<code>rotationStr</code>	Specifies the counterclockwise rotation value in degrees (0, 90, 180, or 270).
<code>mirrorStr</code>	Specifies the direction, if any, in which to flip the cell instance (no, X, or Y).
<code>height</code>	Specifies the size of the text, in user units.
<code>placement</code>	Specifies the x/y-coordinate pair representing the location of the cell instance origin.
<code>testStr</code>	Specifies the text value to give to the new text object.
<code>textId</code>	Specifies the database identifier of the new text object.

Return

<code>MWXSuccess</code>	Occurs if the text is successfully created.
<code>MWXNoLibraryOpened</code>	Occurs if a library is not open.
<code>MWXFail</code>	Occurs if the text is not successfully created for any other reason.

Example

```
char *layerName = "M2R";
char *justStr = "ll";
char *rotationStr = "0";
char *mirrorStr = "no";
int32 height = 3;
```

```

MWXPoint_t *origin;
char *textStr = "Full_Adder";
MWXObjId_t textId;
int rc;

placement->x = 5;
placement->y = 5;
rc = MWXDb_Create_Text (cellId, layerName, justStr, rotationStr, mirrorStr,
height, &origin, textStr, &textId);
switch (rc) {
    case MWXSuccess:
        MWX_Printf("Text '%s' has been created\n",textStr);
        return (textId);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to create text\n");
        return (MWXFail);
}

```

Scheme Equivalent

dbCreateText

(see Physical Implementation Online Help)

MWXDb_Create_Wire_Master

Creates a wire master in the cell identified by `cellId`.

Syntax

```

int MWXDb_Create_Wire_Master(
    MWXCellId_t cellId, /* INPUT */
    int32 wireWidth, /* INPUT */
    int32 pathType, /* INPUT */
    int32 layerType, /* INPUT */
    int32 layerNumber, /* INPUT */
    uchar *wireMasterIndex /* OUTPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell in which to create the wire master.
wireWidth	Specifies the width of the wire, in database units.
pathType	Specifies the path type. 0 means that the path edge aligns with the coordinate; 2 means that the path edge extends beyond the coordinate.
layerType	Represents a 0 (this field is not currently used).
layerNumber	Specifies the layer number for instances of this wire master.
wireMasterIndex	Specifies the internal identifier for the new wire master.

Return

MWXSucces	Occurs if the wire master is successfully created.
MWXFail	Occurs if the wire master is not successfully created.

Example

```
int32 wireWidth = 3;
int32 pathType = 0;
int32 layerType = 0;
int32 layerNumber = 5;
int rc;
uchar wireMasterIndex;
rc = MWXDb_Create_Wire_Master (cellId, wireWidth, pathType, layerType,
layerNumber, &wireMasterIndex);
switch (rc) {
    case MWXSucces:
        MWX_Printf("Wire master is successfully created\n");
        return (MWXSucces);
    default:
        MWX_Printf("ERROR -- Failed to create wire
        master\n");
        return (MWXFail);
}
```

Scheme Equivalent

dbCreateWireMaster

(see Physical Implementation Online Help)

MWXdB_Delete_Cell

Removes a cell from the currently open library. The cell must be closed for the this function to successfully operate.

Syntax

```
int MWXdB_Delete_Cell (  
    char *cellName /* INPUT */  
)
```

Parameters

cellName	Specifies the name of the cell to remove from the library.

Return

MWXSucces	Occurs if the delete is successfully completed.
MWXNoLibraryOpened	Occurs if a library is not currently open.
MWXFail	Occurs if the operation fails for any other reason.

Example

```
char *cellName = "testCell.CEL;1";  
rc = MWXdB_Delete_Cell (cellName);  
    if (rc == MWXSucces) {  
        MWX_Printf("Cell %s is deleted successfully\n",cellName);  
        return (MWXSucces);  
    } else {  
        return (MWXFail);  
    }
```

Scheme Equivalent

```
dbDeleteCell  
(see Physical Implementation Online Help)
```

MWXdB_Delete_CellAttachFile

Deletes an attached file from the specified cell in an opened library.

Syntax

```
int MWXdB_Delete_CellAttachFile (  
    MWXCellId_t cellId, /* INPUT */  
    char *pseudoAccessName /* INPUT */  
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell from which an attached file will be deleted.
<code>pseudoAccessName</code>	Identifies the attached file to be deleted.

Return

<code>MWXNoLibraryOpened</code>	Occurs if no library is open.
<code>MWXFail</code>	Occurs if the attached file is not successfully deleted.
<code>MWXSuccess</code>	Occurs if the attached file is successfully deleted.

Example

```
char *pseudoName = "cell_info";
rc = MWXDb_Delete_CellAttachFile ( cellId, pseudoName );
    switch (rc) {
        case MWXSuccess:
            MWX_Printf("Successfully remove the cell attached file\n");
            return (MWXSuccess);
        case MWXFail:
            MWX_Printf("ERROR -- Failed to delete
            CellAttachFile\n");
            return (MWXFail);
        case MWXNoLibraryOpened:
            MWX_Printf("ERROR -- No open library was
            found\n");
            return (MWXFail);
        default:
            MWX_Printf("ERROR -- unknown return code from
            MWXDb_Delete_CellAttachFile\n");
            return (MWXFail);
    }
```

Scheme Equivalent

`dbDeleteCellAttachedFile`
(see Physical Implementation Online Help)

MWXDb_Delete_CellProperty_ByName

Deletes the property with the given name from the cell identified by the `cellId` database identifier.

Syntax

```
int MWXDb_Delete_CellProperty_ByName (  
    MWXCellId_t cellId, /* INPUT */  
    char *propertyName /* INPUT */  
)
```

Parameters

cellId Represents the database identifier of the cell from which to delete the property.

propertyName Specifies the name of the property to delete.

Return

MWXSuccess Occurs if the property is successfully deleted.

MWXFail Occurs if the property deletion fails.

Example

```
char *propertyName = "original cellInfo";  
if (MWXDb_Delete_CellProperty_ByName (cellId, propertyName) != MWXSuccess) {  
    MWX_Printf("ERROR -- Failed to delete cell property  
        '%s'\n",propertyName);  
    return (MWXSuccess);  
} else {  
    return (MWXFail);  
}  
}
```

MWXDb_Delete_LibAttachFile

Deletes an attached file from an opened library.

Syntax

```
int MWXDb_Delete_LibAttachFile (  
    char *pseudoAccessName /* INPUT */  
)
```

Parameters

`pseudoAccessName` Identifies the attached file to be deleted.

Return

`LibraryOpened` Occurs if no library is open.

`MWXFail` Occurs if the attached file is not successfully deleted.

`MWXSuccess` Occurs if the attached file is successfully deleted.

Example

```
char *pseudoAccessName = "myInfo";
rc = MWXDb_Delete_LibAttachFile ( fileName );

switch (rc) {
    case MWXSuccess:
        MWX_Printf("Library attached file is deleted\n");
        return (MWXSuccess);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to delete
                  LibAttachFile\n");
        return (MWXFail);
    case MWXNoLibraryOpened:
        MWX_Printf("ERROR -- No open library was
                  found\n");
        return (MWXFail);
    default:
        MWX_Printf("ERROR -- unknown return code from
                  MWXDb_Delete_LibAttachFile\n");
        return (MWXFail);
}
```

Scheme Equivalent

`dbDeleteLibAttachedFile`

(see Physical Implementation Online Help)

MWXDb_Delete_Object

Deletes the object identified by `objectId` from the cell identified by `cellId`.

Syntax

```
int MWXDb_Delete_Object(
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t objectId /* INPUT */
)
```


Parameters

cellId	Specifies the database identifier of the cell from which to delete the object.
objectId	Specifies the database identifier of the object to delete.

Return

MWXSuccess	Occurs if the object is successfully deleted.
MWXFail	Occurs if the object is not successfully deleted.

Example

```
rc = MWXDb_Delete_Object (cellId, objectId);
switch (rc) {
    case MWXSuccess:
        return (MWXSuccess);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to delete
                  object\n");
        return (MWXFail);
}
```

Scheme Equivalent

dbDeleteObject (see the *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Delete_ObjectArray

Deletes all `objectCount` objects in the array identified by the `objIdArray` from the cell identified by `cellId`.

Syntax

```
int MWXDb_Delete_ObjectArray(
    MWXCellId_t cellId, /* INPUT */
    int objectCount, /* INPUT */
    MWXObjId_t *objIdArray /* INPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell from which to delete the object array.
<code>objectCount</code>	Specifies the database identifier of the total number of objects in the array.
<code>objIdArray</code>	Specifies the database identifier of the object array.

Return

<code>MWXSuccess</code>	Occurs if the object array is successfully deleted.
<code>MWXFail</code>	Occurs if the object array is not successfully deleted.

Example

```
rc = MWXDb_Delete_ObjectArray (cellId, objCount, objIdArray);
switch (rc) {
    case MWXSuccess:
        return (MWXSuccess);
    case MWXFail:
        MWX_Printf("ERROR -- Fail to delete
                  objects by layer\n");
        return (MWXFail);
}
```

MWXDb_Delete_Objects_ByLayer

Deletes all the objects on a layer, identified layer by layer, in the library identified as `libId`.

Syntax

```
int MWXDb_Delete_Objects_ByLayer(
    MWXLibId_t libId, /* INPUT */
    int layer, /* INPUT */
)
```

Parameters

libId	Specifies the database identifier of the library from which to delete the objects.
layer	Specifies the database identifier of the layer from which to delete the objects.

Return

MWXSuccess	Occurs if the objects on a specific layer are successfully deleted.
MWXFail	Occurs if the objects on a specific layer are not successfully deleted.

Example

```
char *layerName;
int layerNumber;
int rc;

if (MWXDb_Get_LayerNumber_FromName (layerName, &layerNumber) != MWXSuccess) {
    MWX_Printf("Fail to get layer number of layer %s\n",layerName);
    return (MWXFail);
}
rc = MWXDb_Delete_Objects_ByLayer (libId, layerNumber);
switch (rc) {
    case MWXSuccess:
        return (MWXSuccess);
    case MWXFail:
        MWX_Printf(("ERROR -- Fail to delete
                    objects by layer\n");
        return (MWXFail);
}
```

MWXDb_Delete_Pin

Deletes the pin identified by `pinId` from the cell identified by `cellId`.

Syntax

```
int MWXDb_Delete_Pin(
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t pinId /* INPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell from which to delete the pin.
<code>pinId</code>	Specifies the database identifier of the pin to delete.

Return

<code>MWXSuccess</code>	Occurs if the pin is successfully deleted.
<code>MWXFail</code>	Occurs if the pin is not successfully deleted.

Example

```
char *pinName = "IN1";
int rc;

rc = MWXDb_Get_Pin_ByName (cellId, pinName, &pinId);
if (rc != MWXSuccess) {
    MWX_Printf("Fail to get pinId of pin %s\n",pinName);
    return(MWXFail);
}
rc = MWXDb_Delete_Pin (cellId, pinId);
if (rc != MWXSuccess) {
    MWX_Printf("Error -- Failed to delete pin\n");
    return (MWXFail);
}
return (MWXSuccess);
}
```

Scheme Equivalent

`dbDeletePin`
(see Physical Implementation Online Help)

MWXDb_Delete_Property

Deletes the property identified by `propId` from the cell identified by `cellId`.

Syntax

```
int MWXDb_Delete_Property (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t propId /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell from which to delete the property.
propId	Specifies the database identifier of the property to delete.

Return

MWXSuccess	Occurs if the property is successfully deleted.
MWXFail	Occurs if the property is not successfully deleted.

Example

```
rc = MWXDb_Delete_Property (cellId, propId);
switch (rc) {
    case MWXSuccess:
        MWX_Printf("Successfully delete property\n");
        return (MWXSuccess);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to delete
                    property\n");
        return (MWXFail);
}
```

Scheme Equivalent

dbDeleteProperty

(see Physical Implementation Online Help)

MWXDb_Delete_Property_ByName

Deletes the property with the name `propertyName` from the object identified by `objId` contained in the cell identified by `cellId`.

Syntax

```
int MWXDb_Delete_Property_ByName (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t objId, /* INPUT */
    char *propertyName /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell containing the object from which to delete the property.
objId	Specifies the database identifier of the object from which to delete the property.
propertyName	Specifies the name of the property to delete.

Return

MWXSuccess	Occurs if the property is successfully deleted.
MWXFail	Occurs if the property is not successfully deleted.

Example

```
MWXObjId_t pinId = 1245;
char *propertyName = "original Net";

int rc;
rc = MWXDb_Delete_Property_ByName (cellId, objId, propertyName);
switch (rc) {
    case MWXSuccess:
        MWX_Printf("Property %s is deleted\n",propertyName);
        return (MWXSuccess);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to delete
                    property\n");
        return (MWXFail);
}
```

Scheme Equivalent

dbDeletePropertyByName
(see Physical Implementation Online Help)

MWXDb_Delete_RefLib_ByName

Deletes a reference library from an opening library.

Syntax

```
int MWXDb_Delete_RefLib_ByName (
    char *refLibraryName /* INPUT */
)
```

Parameters

<code>refLibraryName</code>	Name of the reference library to delete from the opening library.

Return

<code>MWXNoLibraryOpened</code>	Occurs if the library is not open.
<code>MWXSuccess</code>	Occurs if the reference library is removed from the library.
<code>MWXFail</code>	Occurs if the reference library cannot be removed from the library.

Example

```
char *refLibraryName = "macro_library";
rc = MWXDb_Delete_RefLib_ByName ( refLibraryName )

switch (rc) {
    case MWXSuccess:
        MWX_Printf("Reference Library %s is successfully deleted
                    \n",refLibraryName);
        return (MWXSuccess);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to operate. \n");
        return (MWXFail);
}
```

Scheme Equivalent

`dbDeleteRefLibByName`

(see Physical Implementation Online Help)

MWXDb_Disconnect

Disconnects the object identified by `objectId` from the net identified by `netId`.

Syntax

```
int MWXDb_Disconnect(
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t netId, /* INPUT */
    MWXObjId_t objectId /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell containing the net and the object to disconnect.
netId	Specifies the database identifier of the net from which the object is to be disconnected.
objectId	Specifies the database identifier of the object to disconnect from the net.

Return

MWXSuccess	Occurs if Milkyway successfully disconnects the object from the net.
MWXFail	Occurs if Milkyway fails to successfully disconnect the object from the net.

Example

```
MWXObjId_t netId = 1005;
MWXObjId_t pinId = 1245;
int rc;
rc = MWXDb_Disconnect (cellId, netId, pinId);
switch (rc) {
    case MWXSuccess:
        return (MWXSuccess);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to disconnect object
                    from net\n");
        return (MWXFail);
}
```

Scheme Equivalent

dbDisconnect

(see Physical Implementation Online Help)

MWXDb_Disconnect_PortInstance_ByName

Disconnects the specified port instance from the net identified by netId.

Syntax

```
int MWXDb_Disconnect_PortInstance_ByName (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t netId, /* INPUT */
    char *cellInstanceName, /* INPUT */
    char *portInstanceName /* INPUT */
)
```


Parameters

<code>cellId</code>	Specifies the database identifier of the cell containing the net and the port instance to disconnect.
<code>netId</code>	Specifies the database identifier of the net from which the port instance is to be disconnected.
<code>cellInstanceName</code>	Specifies the name of the cell instance containing the port instance to be disconnected.
<code>portInstanceName</code>	Specifies the name of the port instance to be disconnected.

Return

<code>MWXSuccess</code>	Occurs if Milkyway successfully disconnects the port instance from the net.
<code>MWXFail</code>	Occurs if Milkyway fails to successfully disconnect the port instance from the net.

Example

```
rc = MWXDb_Disconnect_PortInstance_ByName (
    cellId, netId, cellInstanceName, portInstanceName);
switch (rc) {
    case MWXSuccess:
        return (MWXSuccess);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to disconnect port to
            net\n");
        return (MWXFail);
}
```

Scheme Equivalent

`dbDisconnectPortInstByName`
(see Physical Implementation Online Help)

MWXDb_Display_DiodePinInstMaster_FromDiodePortInstMaster

Displays the information for all DiodePinInstMasters from DiodePortInstMasters contained in the cell identified as `cellId`.

Syntax

```
int MWXDb_Display_DiodePinInstMaster_FromDiodePort-Instmaster(  
    MWXCellId_t cellId /* INPUT : cell identification  
        number. */  
)
```

Parameters

cellId Specifies the database identifier of the cell containing DiodePinInstMasters.

Return

MWXSuccess Occurs when the information is successfully displayed.

MWXFail Occurs when the information is not successfully displayed.

Example

```
rc = MWXDb_Display_DiodePinInstMaster_FromDiodePortInstMaster(cellId);  
    switch (rc) {  
        case MWXSuccess:  
            return (MWXSuccess);  
        case MWXFail:  
            MWX_Printf("Fail to display the information  
                for all DiodePinInstMasters from  
                DiodePortInstMaster . \n");  
            return (MWXFail);  
        default:  
            return (MWXFail);  
    }
```

MWXDb_Display_DiodePortInst

Displays the information for all DiodePortInstances contained in the cell identified as cellId.

Syntax

```
int MWXDb_Display_DiodePortInst(  
    MWXCellId_t cellId /* INPUT : cell identification number. */  
)
```

Parameters

`cellId` Specifies the database identifier of the cell containing DiodePortInstances.

Return

`MWXSuccess` Occurs when the information is successfully displayed.

`MWXFail` Occurs when the information is not successfully displayed.

Example

```
rc = MWXDb_Display_DiodePortInst(cellId);
    switch (rc) {
        case MWXSuccess:
            return (MWXSuccess);
        case MWXFail:
            MWX_Printf("Fail to display the information
for all DiodePortInstances. \n");
            return (MWXFail);
        default:
            return (MWXFail);
    }
```

MWXDb_Display_DiodePortInst_FromCellInst

Displays the information for all DiodePortInstances from CellInstance contained in the cell identified as `cellId`.

Syntax

```
int MWXDb_Display_DiodePortInst_FromCellInst(
    MWXCellId_t cellId /* INPUT : cell identification number. */
)
```

Parameters

`cellId` Specifies the database identifier of the cell containing DiodePortInstances.

Return

`MWXSuccess` Occurs when the information is successfully displayed.

`MWXFail` Occurs when the information is not successfully displayed.

Example

```
rc =MWXDb_Display_DiodePortInst_FromCellInst(cellId);
    switch (rc) {
        case MWXSuccess:
            return (MWXSuccess);
        case MWXFail:
            MWX_Printf("Fail to display the information
                for all DiodePortInstances from CellInstance.
                \n");
            return (MWXFail);
        default:
            return (MWXFail);
    }
```

MWXDb_Display_DiodePortInst_FromNet

Displays the information for all DiodePortInstances from nets contained in the cell identified as `cellId`.

Syntax

```
int MWXDb_Display_DiodePortInst_FromNet(
    MWXCellId_t cellId /* INPUT : cell identification number. */
)
```

Parameters

`cellId` Specifies the database identifier of the cell containing DiodePortInstances.

Return

`MWXSuccess` Occurs when the information is successfully displayed.

`MWXFail` Occurs when the information is not successfully displayed.

Example

```
rc = MWXDb_Display_DiodePortInst_FromNet(cellId);
    switch (rc) {
        case MWXSuccess:
            return (MWXSuccess);
        case MWXFail:
            MWX_Printf("Fail to display the information
            for all DiodePortInstances from Net. \n");
            return (MWXFail);
        default:
            return (MWXFail);
    }
```

MWXDb_Display_DiodePortInstMaster

Displays the information for all DiodePortInstMasters contained in the cell identified as `cellId`.

Syntax

```
int MWXDb_Display_DiodePortInstMaster(
    MWXCellId_t cellId /* INPUT : cell identification number. */
)
```

Parameters

`cellId` Specifies the database identifier of the cell containing DiodePortInstMasters.

Return

`MWXSuccess` Occurs when the information is successfully displayed.

`MWXFail` Occurs when the information is not successfully displayed.

Example

```
rc = MWXDb_Display_DiodePortInstMaster(cellId);
    switch (rc) {
        case MWXSuccess:
            return (MWXSuccess);
        case MWXFail:
            MWX_Printf("Fail to display the information
            for all DiodePortInstMasters. \n");
            return (MWXFail);
        default:
            return (MWXFail);
    }
```

MWXDb_Display_DiodePortInstMaster_FromDiodeCellInstMaster

Displays the information for all DiodePortInstMasters from DiodeCellInstMaster contained in the cell identified by `cellId`.

Syntax

```
int MWXDb_Display_DiodePortInstMaster_FromDiodeCellInstMaster(
    MWXCellId_t cellId /* INPUT : cell identification number. */
)
```

Parameters

`cellId` Specifies the database identifier of the cell containing DiodePortInstMasters.

Return

`MWXSuccess` Occurs when the information is successfully displayed.

`MWXFail` Occurs when the information is not successfully displayed.

Example

```
rc = MWXDb_Display_DiodePortInstMaster_FromDiodeCellInstMaster(cellId);
    switch (rc) {
        case MWXSuccess:
            return (MWXSuccess);
        case MWXFail:
            MWX_Printf("Fail to display the information
for all DiodePortInstMasters from Diode CellInstMaster. \n");
            return (MWXFail);
        default:
            return (MWXFail);
    }
```

MWXDb_Dump_TechFile

Dumps the technology file information into the specified file.

Syntax

```
int32 MWXDb_Dump_TechFile (
    MWXLibId_t libId /* INPUT */
    char *fileName /* INPUT */
)
```

Parameters

libId	Specifies the ID of the library to be queried.
fileName	Specifies the file into which the tech file information is dumped.

Return

MWXSuccess	Occurs if the tech file information is successfully dumped.
MWXFail	Occurs if the tech file information is not successfully dumped.

Dynamic Example

```
MWXSCM_t mwDumpTechFile(MWXSCM_t list) {  
  
    MWXLibId_t libId;  
    char filename[64];  
    int rc;  
  
    if (MWXSc_Scanf(list, "os", &libId, filename) != 2) {  
        MWX_Printf("ERROR - Usage: mwDumpTechFile libId techFileName.\n");  
        return MWX_FALSE_SCM;  
    }  
  
    rc = MWXDb_Dump_TechFile(libId, filename);  
  
    switch (rc) {  
    case MWXSuccess:  
        MWX_Printf("Techfile has been dumped to %s\n", filename);  
        return MWX_TRUE_SCM;  
    case MWXFail:  
        MWX_Printf("Failed to dump techfile\n");  
        return MWX_FALSE_SCM;  
    default:  
        MWX_Printf("ERROR -- Unknown return code %d. MWXDb_Dump_TechFile "  
            "failed\n", rc);  
        return MWX_FALSE_SCM;  
    }  
}
```

Stand-Alone Example

```
int DumpTechFile(MWXLibId_t libId, char *filename) {  
  
    int rc;  
  
    MWX_Printf("INFO -- MWXDb_Dump_TechFile\n");  
    rc = MWXDb_Dump_TechFile(libId, filename);  
  
    switch (rc) {  
    case MWXSuccess:
```



```

    MWX_Printf("Techfile has been dumped to %s\n", filename);
    return MWXSuccess;
case MWXFail:
    MWX_Printf("Failed to dump techfile\n");
    return MWXFail;
default:
    MWX_Printf("ERROR -- Unknown return code %d. MWXDb_Dump_TechFile "
              "failed\n", rc);
    return MWXFail;
}
}

```

Scheme Equivalent

dbDumpTechFile

See Also

MWXDb_Replace_TechInfo

MWXDb_End_ReadLib

Unlocks specified library.

Syntax

```

int32 MWXDb_End_ReadLib (
    MWXLibId_t libId /* INPUT */
)

```

Parameters

libId	Specifies the ID of the library to unlock.

Return

MWXSuccess	Occurs if the library write lock is successfully removed.

MWXNoLibraryOpened	Occurs if the design library is not open.

MWXFail	Occurs if the library write lock is not removed.

Dynamic Example

```

MWXSCM_t mwEndWriteLib(MWXSCM_t list) {

    MWXLibId_t libId;
    int rc;

    if (MWXSc_Scanf(list, "o", &libId) != 1) {
        MWX_Printf("ERROR -- mwEndWriteLib requires 1 argument.\n");
        return MWX_FALSE_SCM;
    }
}

```

```

rc = MWXDb_End_WriteLib(libId);

switch (rc) {
case MWXSuccess:
    MWX_Printf("Successfully removed write lock.\n");
    return MWX_TRUE_SCM;
case MWXNoLibraryOpened:
    MWX_Printf("ERROR -- No Library opened. MWXDb_End_WriteLib "
               "failed.\n");
    return MWX_FALSE_SCM;
case MWXFail:
    MWX_Printf("Unable to remove write lock.\n");
    return MWX_FALSE_SCM;
default:
    MWX_Printf("ERROR -- Unknown return code %d. "
               "MWXDb_End_WriteLib failed.\n", rc);
    return MWX_FALSE_SCM;
}

```

Stand-Alone Example

```

int MWXDb_End_ReadLib(MWXLibId_t libId) {

    int rc;

    MWX_Printf("INFO -- MWXDb_End_ReadLib.\n");
    rc = MWXDb_End_ReadLib(libId);

    switch (rc) {
case MWXSuccess:
    MWX_Printf("Successfully removed read lock.\n");
    return MWXSuccess;
case MWXNoLibraryOpened:
    MWX_Printf("ERROR -- No Library opened. MWXDb_End_ReadLib "
               "failed.\n");
    return MWXFail;
case MWXFail:
    MWX_Printf("Unable to remove read lock.\n");
    return MWXFail;
default:
    MWX_Printf("ERROR -- Unknown return code %d. "
               "MWXDb_End_ReadLib failed.\n", rc);
    return MWXFail;
}
}

```

See Also

[MWXDb_Begin_ReadLib](#)

MWXDb_End_WriteLib

To end the library write lock. Performs an update of the library lib file to disk.

Syntax

```
int MWXDb_End_WriteLib(  
MWXLibId_t libId /* INPUT */  
)
```

Parameters

libId Specifies the database identifier of the library.

Return

MWXSuccess Occurs if successfully end the lock library for write

MWXFail Occurs if fail to end the lock library for write

MWXNoLibraryOpened Occurs if there is no library currently opens

Example

```
int MWXDb_End_WriteLib(MWXLibId_t libId) {  
    int rc;  
  
    MWX_Printf("INFO -- MWXDb_End_WriteLib.\n");  
    rc = MWXDb_End_WriteLib(libId);  
  
    switch (rc) {  
    case MWXSuccess:  
        MWX_Printf("Successfully removed write lock.\n");  
        return MWXSuccess;  
    case MWXNoLibraryOpened:  
        MWX_Printf("ERROR -- No Library opened. MWXDb_End_WriteLib "  
            "failed.\n");  
        return MWXFail;  
    case MWXFail:  
        MWX_Printf("Unable to remove write lock.\n");  
        return MWXFail;  
    default:  
        MWX_Printf("ERROR -- Unknown return code %d. "  
            "MWXDb_End_WriteLib failed.\n", rc);  
        return MWXFail;  
    }  
}
```

See Also

[MWXDb_Begin_WriteLib](#)

MWXdB_Fetch_Object

This command retrieves all the field names and the data contained within these fields for a given object within a cell.

Syntax

```
int MWXdB_Fetch_Object (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t objId, /* INPUT */
    int *fieldCount, /* OUTPUT */
    MWXCharList **fieldsArray /* OUTPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell from which to get the field names and current values.
<code>objId</code>	Specifies the database identifier of the object from which the field names and current values are being retrieved.
<code>fieldCount</code>	Number of fields returned in this object.
<code>fieldsArray</code>	An array of object field names and the data that is contained within these fields for the given object. It is your responsibility to free the memory.

Return

<code>MWXSuccess</code>	Occurs if object field names and data are successfully retrieved.
<code>MWXFail</code>	Occurs if object field names and data are not successfully retrieved.

Example

```
rc = MWXdB_Fetch_Object(dbId, objId, &fieldCount, &fieldsArray);
switch (rc) {
    case MWXSuccess:
        p = fieldsArray;
        for (i=0;i<fieldCount;i++) {
            MWX_Printf("%s\n", (p->name));
            p = p->next;
        }
        while (fieldsArray) {
            p = fieldsArray->next;
            free (fieldsArray->name);
            free (fieldsArray);
            fieldsArray = p;
        }
        return (MWXSuccess);
}
```

```

        case MWXFail:
            MWX_Printf("\nERROR -- Failed to fetch object field name
            and data\n");
            return (MWXFail);
        case MWXNoLicense:
            MWX_Printf("\nERROR -- No license to run this
            function!\n");
            return (MWXFail);
        default:
            MWX_Printf("\nERROR -- unknown return code from
            MWXDb_Fetch_Object\n");
            return (MWXFail);
    }

```

Scheme Equivalent

dbFetchObject

(see Physical Implementation Online Help)

MWXDb_Get_AccessEdgeFromPin

Retrieves the AccessEdge record from a pin. It is your responsibility to recover the memory that is allocated for the ObjArray object when the data is no longer needed.

Syntax

```

int MWXDb_Get_AccessEdgeFromPin (
    MWXCellId_t cellId, /*INPUT*/
    MWXObjId_t pinId, /*INPUT: only get the object IDs
                       attached to this pin id */
    int32 *objectCount, /*OUTPUT: number of members */
    MWXObjId_t **objectsIdArray /*OUTPUT: an array of
    all member id. It is the caller's responsibility
    to free the memory, i.e., free (*memberIdArray). */
)

```

Parameters

<code>cellId</code>	Specifies the database identifier of a cell.
<code>pinId</code>	Specifies the database identifier of a pin.
<code>objectCount</code>	Specifies the number of the objects being returned in <code>objectsIdArray</code> .
<code>objectsIdArray</code>	Specifies the array of database identifiers for objects retrieved.

Return

<code>MWXSuccess</code>	Occurs if objects are successfully retrieved.
<code>MWXFail</code>	Occurs if objects are not successfully retrieved.

Dynamic Example

```
rc = MWXDb_Get_AccessEdgeFromPin (cellId, pinId, &objectCount, &objectsIdArray);
    MWX_Printf("INFO -- rc=%d\n",rc);
    switch (rc) {
    case MWXSuccess:
        for (i = 0; i < objectCount; i++) {
            sprintf(buf, "(define objects (append objects '(%d)))",
                objectsIdArray[i]);
            MWXSc_Evaluate (buf);
        }
    if (objectCount > 0) free (objectsIdArray);
        return (MWXSc_Make_IntegerSCM (objectCount));
    case MWXFail:
        MWX_Printf("ERROR -- Failed to get AccessEdge IDs\n");
        return (MWX_FALSE_SCM);
    default:
        MWX_Printf("ERROR -- unknown return code from
MWXDb_Get_AccessEdgeFromPin\n");
        return (MWX_FALSE_SCM);
    }
```

MWXDb_Get_AllCellProperties

Gets the database identifiers of all of the properties attached to the cell identified by `cellId`. Note that `MWXDb_Get_AllCellProperties` allocates memory for the array. It is your responsibility to recover this memory when the data is no longer needed.

Syntax

```
int MWXDb_Get_AllCellProperties(  
    MWXCellId_t cellId, /* INPUT */  
    int *propertyCount, /* OUTPUT */  
    MWXObjId_t **propertiesIdArray /* OUTPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell from which to get properties.
propertyCount	Specifies the number of the properties retrieved.
propertiesIdArray	Specifies the array of the property database identifiers.

Return

MWXSuccess	Occurs if the properties are successfully retrieved.
MWXFail	Occurs if the properties are not successfully retrieved (note that an attempt to retrieve properties from a cell with no properties is not a failure but will simply return a count of 0 with an empty array).

Example

```
rc = MWXDb_Get_AllCellProperties (cellId,  
                                &propertyCount,&propertiesIdArray);  
switch (rc) {  
    case MWXSuccess:  
        break;  
    case MWXFail:  
        MWX_Printf("ERROR -- failed to get  
                  properties\n");  
        break;  
}
```

MWXDb_Get_AllObjectProperties

Gets the database identifiers of all the properties attached to the object identified by objectId. The MWXDb_Get_AllCellProperties function allocates memory for the array. It is your responsibility to recover this memory when you no longer need the data.

Syntax

```
int MWXDb_Get_AllObjectProperties(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t objectId, /* INPUT */
```

```

        int *propertyCount, /* OUTPUT */
        MWXObjId_t **propertiesIdArray /* OUTPUT */
    )

```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell containing the object from which to get the properties.
<code>objectId</code>	Specifies the database identifier of the object from which to get the properties.
<code>propertyCount</code>	Specifies the number of the properties retrieved.
<code>propertiesIdArray</code>	Specifies the array of the property database identifiers.

Return

<code>MWXSuccess</code>	Occurs if the properties are successfully retrieved.
<code>MWXFail</code>	Occurs if the properties are not successfully retrieved (note that an attempt to retrieve properties from an object with no properties is not a failure but simply returns a count of 0 with an empty array).

Example

```

rc = MWXDb_Get_AllObjectProperties (cellId,
                                   objectId, &propertyCount,&propertiesIdArray);
switch (rc) {
    case MWXSuccess:
        break;
    case MWXFail:
        MWX_Printf("ERROR -- failed to get
                  properties\n");
        break;
}

```

MWXDb_Get_AllPin_ByName

Gets the database identifiers of all the pins in the cell identified by `cellId` with the name supplied in `pinName`. It is your responsibility to recover the memory allocated for the `PinIdArray` object when the data is no longer needed.

Syntax

```

int MWXDb_Get_AllPin_ByName(
    MWXCellId_t cellId, /* INPUT */
    char *pinName,      /* INPUT */

```



```

        int32 *numberOfPin, /* INPUT(OUTPUT) *
        MWXObjId_t **pinIdArray /* OUTPUT */
    )

```

Parameters

cellId	Specifies the database identifier of the cell from which to get the pins.
pinName	Specifies the names of the pins to get.
numberOfPin	Specifies, as input, the maximum number of pins to retrieve, and as output, the actual number of pins retrieved.
pinIdArray	Specifies the array of pin database identifiers.

Return

MWXSuccess	Occurs if the pins are successfully retrieved.
MWXFail	Occurs if the pins are not successfully retrieved.

Example

```

rc = MWXDb_Get_AllPin_ByName (cellId, pinName, &numberOfPin, &pinIdArray);
switch (rc) {
    case MWXSuccess:
        break;
    case MWXFail:
        MWX_Printf("ERROR -- failed to get pins\n");
        break;
}

```

Scheme Equivalent

dbGetPinByName (see *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Get_CaseSensitive

This function gets the case sensitive mode.

Syntax

```

int MWXDb_Get_CaseSensitive(
int *mode /* OUTPUT */
)

```

Parameters

mode Case sensitive mode

Return

MWXSucces Occurs if successfully gets the case sensitive mode

MWXFail Occurs if fail to get the case sensitive mode

Example

```
rc = MWXDb_Get_CaseSensitive(&mode);
switch (rc) {
    case MWXSucces:
        return (mode);
    case MWXFail:
        printf("Fail to get the case sensitive mode\n");
        return 0;
    Default:
        printf("Unknown return code \n");
        return 0;
}
```

MWXDb_Get_Cell_AccessMode

Gets the current cell access mode, such as read only or write mode.

Syntax

```
int MWXDb_Get_Cell_AccessMode (
    MWXCellId_t cellId, /* INPUT */
    char *accessMode /* OUTPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell.
<code>accessMode</code>	Specifies the access mode, which can be <code>w</code> or <code>r</code> .

Return

<code>MWXSuccess</code>	Occurs if Milkyway successfully gets the access mode.
<code>MWXFail</code>	Occurs if Milkyway cannot successfully get the access mode.

Example

```
if (MWXDb_Get_Cell_AccessMode(cellId,&accessMode) != MWXSuccess) {  
    MWX_Printf("Fail to get Cell access mode \n");  
    return (MWXFail);  
}  
  
    printf ("Cell AccessMode is %s\n",accessMode);
```

MWXDb_Get_CellAttachedFiles

Returns a list of attached files for the specified cell in the specified library.

Syntax

```
public int  
    MWXDb_Get_CellAttachedFiles(  
        MWXLibId_t libId /* INPUT */  
        char *cellName /* INPUT */  
        char *viewName /* INPUT */  
        int32 version /* INPUT */  
        int *attachedFileCnt /* OUTPUT */  
        MWXDBString_t **attachedFiles /* OUTPUT */  
        MWXDBString_t **pseudoAccessNames /* OUTPUT */  
        int32 **fileExistStatus /* OUTPUT */  
    )
```

Parameters

<code>libId</code>	Specifies the design library database identification.
<code>cellName</code>	Specifies the name of the cell to receive the list of attached files.
<code>viewName</code>	Specifies the view name of the cell to receive the list of attached files.
<code>version</code>	Specifies the cell version. Use 0 for the latest version.
<code>attachedFileCnt</code>	Returned value for the total number of attached files in the cell.
<code>attachedFiles</code>	Specifies the name list of all attached files in the cell.
<code>pseudoAccessNames</code>	Specifies the name list of all pseudo access names in the cell.
<code>fileExistStatus</code>	Specifies the list of the file existence flag.

Return

<code>MWXSuccess</code>	Occurs when the attached file information for the specified cell is successfully retrieved.
<code>MWXFail</code>	Occurs when Milkyway fails to retrieve the attached file information for the specified cell.

Description

The number of attached files is stored in "attachedFileCnt." The point of pointer of file names is stored in "attachedFiles." The point of pointer of pseudo names is stored in "pseudoAccessNames."

For a cell of CEL view, the attached file name string appears similar to the following:

```
"CEL/killUnit:2_1"
```

For a cell of LOGIC/LM view, the attached file name string appears similar to the following:

```
"LOGIC/design_1.db"
```

Note:

It is your responsibility to free the allocated memory of the string pointer.

Example

```
void free_string_memory (MWXDBString_t *str) {  
    if (str) {
```

```

        free(str);
    }
}
void free_int_memory (int *num) {
    if (num) {
        free(num);
    }
}
void Print_CellAttachedFiles (
    MWXLibId_t libId,
    char      *cellNames,
    char      *viewNames,
    int32     version) {

    int k;
    MWXDBString_t *attachedFiles, *pseudoAccessNames;
    int attachedFileCnt;
    int *fileExistStatus;

    if (MWXDb_Get_CellAttachedFiles(libId, cellNames,
        viewNames, version, &attachedFileCnt,
        &attachedFiles, &pseudoAccessNames,
        &fileExistStatus) != MWXSuccess) {
        MWX_Printf("FAIL - Unable to get the cell
            attachedFiles\n");
        exit (MWXFail);
    }
    if (attachedFileCnt > 0) {
        for (k = 0; k < attachedFileCnt; k++) {
            MWX_Printf("SUCCESS get the attached files.
                Their information are:\n");
            MWX_Printf("pseudoAccessNames \t attachedFiles
                \t fileExistStatus\n");
            MWX_Printf("%s \t %s \t %d\n",
                pseudoAccessNames[k],
                attachedFiles[k], fileExistStatus[k]);
        }
    } else {
        MWX_Printf ("There is no attached file for this
            cell\n");
    }
    free_string_memory(pseudoAccessNames);
    free_string_memory(attachedFiles);
    free_int_memory(fileExistStatus);
}

```

MWXDb_Get_CellBBox

Retrieves the coordinates of the bounding box of a cell.

Syntax

```
int MWXDb_Get_CellBBox(  
    MWXCellId_t cellId, /* INPUT */  
    int32 *left, /* OUTPUT */  
    int32 *bottom, /* OUTPUT */  
    int32 *right, /* OUTPUT */  
    int32 *top, /* OUTPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell for which to retrieve the bounding box.
left	Specifies the x-coordinate for the left side of the bounding box.
bottom	Specifies the y-coordinate for the bottom of the bounding box.
right	Specifies the x-coordinate for the right side of the bounding box.
top	Specifies the y-coordinate for the top of the bounding box.

Return

MWXSuccess	Occurs if coordinates are successfully retrieved.
MWXFail	Occurs if coordinates are not successfully retrieved.

Example

```
rc = MWXDb_Get_CellBBox (cellId, &left, &bottom, &right, &top);  
if (rc == MWXSuccess) {  
    MWX_Printf ("Cell bounding box: (%d,%d),  
                (%d,%d)\n", left,bottom, right, top);}
```

MWXDb_Get_CellBoundary

Retrieves the coordinates of the points defining the boundary of a cell.

Syntax

```
int MWXDb_Get_CellBoundary(  
    MWXCellId_t cellId, /* INPUT */  
    int *numPoint, /* OUTPUT */  
    MWXPoint_t **pointArray /* OUTPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell for which to retrieve the boundary.
numPoint	Specifies the number of points in the point array.
pointArray	Specifies the x/y-coordinate pairs representing the points that define the cell boundary.

Return

MWXSucces	Occurs if coordinates are successfully retrieved.
MWXFail	Occurs if coordinates are not successfully retrieved.

Example

```
rc = MWXDb_Get_CellBoundary (cellId, &numPoint,
                             &pointArray);
if (rc == MWXSucces) {
    for (i = 0; i < numPoint; i++) {
        MWX_Printf("Cell point: (%d,%d)\n",
                   pointArray[i].x,pointArray[i].y);
    }
}
```

MWXDb_Get_CellFullName

Retrieves the full name of a cell in the format "cellName, viewName:version."

Syntax

```
public int
MWXDb_Get_CellFullName(
    MWXLibId_t libId, /* INPUT */
    char *cellName, /* INPUT */
    char *viewName, /* INPUT */
    int32 version, /* INPUT */
    char** cellFileName /* OUTPUT */
)
```

Parameters

libId	Specifies the design library database identification.
cellName	Specifies the name of the cell to receive the full cell name.
viewName	Specifies the view name of the cell to receive the full cell name.
version	Specifies the cell version. Use 0 for the latest version.
cellFileName	Specifies the full name of the cell.

Return

MWXNoLibraryOpened	Occurs when no library opened.
MWXSuccess	Occurs when the full cell name is successfully retrieved.
MWXFail	Occurs when Milkyway fails to retrieve the full cell name.

Example

```
void Get_CellFullName (
    MWXLibId_t libId,
    char      *cellNames,
    char      *viewNames,
    int32     version)
{
    char *cellFullName;
    MWXCellId_t cellId;
    if (MWXDb_Get_CellFullName(libId, cellNames,
                              viewNames, version,
                              &cellFullName) != MWXSuccess) {
        MWX_Printf("FAIL - Unable to get the cell full
                  name\n");
        exit (MWXFail);
    }
    MWX_Printf("Cell Full Name is %s\n", cellFullName );
    if (MWXDb_Open_Cell(cellFullName, 'r', &cellId) !=
        MWXSuccess) {
        MWX_Printf("FAIL - Invalid cellName\n");
        exit (MWXFail);
    }
    MWXDb_Close_Cell_And_NoSave(cellId);
    MWX_Printf("Finish testing the cell full name\n")
}
```



```
}
```

MWXDb_Get_CellFullPath

Retrieves the full physical path name of a specified cell.

Syntax

```
public int
MWXDb_Get_CellFullPath(
    MWXLibId_t libId,           /* INPUT */
    char *cellName,            /* INPUT */
    char *viewName,            /* INPUT */
    int32 version,             /* INPUT */
    char** cellFilePath        /* OUTPUT */
)
```

Parameters

libId	Specifies the design library database identification.
cellName	Specifies the name of the cell to receive the full physical path name.
viewName	Specifies the view name of the cell to receive the full physical path name.
version	Specifies the cell version. Use 0 for the latest version.
cellFilePath	Specifies the full physical path name of the cell.

Return

MWXNoLibraryOpened	Occurs when no library opened.
MWXSuccess	Occurs when the full physical path name of the cell is successfully retrieved.
MWXFail	Occurs when Milkyway fails to retrieve the full physical path name of the cell.

Example

```
void Print_CellFullPath (
    MWXLibId_t libId,
    char *cellName,
    char *viewName,
    int32 version)
{
    char *cellFullPath;
```

```

        MWXCellId_t cellId;

        if (MWXDb_Get_CellFullPath(libId, cellName,
            viewName, version, &cellFullPath) != MWXSuccess) {
            MWX_Printf("FAIL - Unable to get the cell full
                name\n");
            exit (MWXFail);
        }
        MWX_Printf("Cell Full Path is %s\n", cellFullPath );
    }
}

```

MWXDb_Get_CellInfo_cellNameAndView

This function retrieves the data for the cellName and view name of the desired cell.

Syntax

```

int MWXDb_Get_CellInfo_cellNameAndView(
MWXCellId_t  cellId /* INPUT */
char **completeName /* OUTPUT */
)

```

Parameters

cellId	Specifies the cellId in the database
completeName	Complete name including cellName.viewName

Return

MWXSuccess	Occurs if successfully gets the cellName and view Name.
MWXFail	Occurs if fail to get the cellName and view Name

Example

```

rc = MWXDb_Get_CellInfo_cellNameAndView(cellId, &completeName);
switch (rc) {
    case MWXSuccess:
        printf("The complete name of the cell is %s\n", completeName);
        break
    case MWXFail:
        printf("Fail to retrieve the cell and view name of the cell\n");
        break;
    Default:
        printf("Unknown return code \n");
        break;
}

```

See Also

MWXDb_Get_CellInfo_fullName

MWXDb_Get_CellInfo_fullName

This function retrieves the data for the complete cell name of the cellInfo record.

Syntax

```
int MWXDb_Get_CellInfo_fullName(  
MWXCellId_t cellId /* INPUT */  
char **completeName /* OUTPUT */  
)
```

Parameters

cellId	Specifies the cellId in the database
completeName	Full name of the cell

Return

MWXSuccess	Occurs if successfully gets the cell full name
MWXFail	Occurs if fail to get the cell full name

Example

```
rc = MWXDb_Get_CellInfo_fullName(cellId, &completeName);  
switch (rc) {  
    case MWXSuccess:  
        printf("The full name of the cell is %s\n", completeName);  
        break  
    case MWXFail:  
        printf("Fail to retrieve the cell full name\n");  
        break;  
    Default:  
        printf("Unknown return code \n");  
        break;  
}
```

MWXDb_Get_CellInstance_ByName

Retrieves the database identifier for the cell with the instance name specified by cellInstanceName. The database identifier of the cell instance is returned in cellInstanceId. The cellInstanceId is undefined if the return code is not MWXSuccess.

MWXDb_Get_CellInfo_fullName

Syntax

```
int MWXDb_Get_CellInstance_ByName(  
    MWXCellId_t cellId, /* INPUT */  
    char *cellInstanceName, /* INPUT */  
    MWXObjId_t *cellInstanceId /* OUTPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell containing the cell instance.
cellInstanceName	Specifies the instance name of the cell to retrieve.
cellInstanceId	Specifies the database identifier of the cell instance with the name cellInstanceName.

Return

MWXSuccess	Occurs if the net is successfully found.
MWXFail	Occurs if the net is not successfully found.

Example

```
rc = MWXDb_Get_CellInstance_ByName (cellId, cellInstName, &cellInstId);  
  
switch (rc) {  
    case MWXSuccess:  
        break;  
    case MWXFail:  
        MWX_Printf("ERROR -- Failed to get cell  
            instance\n");  
        break;  
}
```

Scheme Equivalent

dbGetCellInstByName
(see Physical Implementation Online Help)

MWXDb_Get_CellInstanceBBox

Retrieves the bounding BBox of the cell Instance.

Syntax

```
int32 MWXDb_Get_CellInstanceBBox(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t cellInstId, /* OUTPUT */
```

```

int32 *left, /* OUTPUT */
int32 *bottom, /* OUTPUT */
int32 *right, /* OUTPUT */
int32 *top /* OUTPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell that was opened.
objId	Specifies the database identifier of the cell instance.
left	Specifies the x-coordinate of the left side of the bounding box.
bottom	Specifies the y-coordinate of the bottom of the bounding box.
right	Specifies the x-coordinate of the right side of the bounding box.
top	Specifies the y-coordinate of the top of the bounding box.

Return

MWXSuccess	Occurs if the cell instance bounding box is successfully retrieved.
MWXFail	Occurs if the cell instance bounding box is not successfully retrieved.

Example

```

rc = MWXDb_Get_CellInstanceBBox (cellId, cellInstId, &left, &bottom, &right, &top);
switch (rc) {
    case MWXSuccess:
        MWX_Printf("CellInstBBox      : (%d,%d,%d,%d) \n", left, bottom, right, top);
        return (MWXSuccess);
        break;
    case MWXFail:
        return (MWXFail);
        break;
    default:
        return (MWXFail);
}
return (rtn);
}

```

MWXDb_Get_CellInstanceOrArray_ByName

Gets the specified cell instance or cell instance array ID.

Syntax

```
int MWXDb_Get_CellInstanceOrArray_ByName (
    MWXCellId_t cellId, /* INPUT */
    char *instName /* INPUT */
    int ignoreErrors, /* INPUT */
    MWXObjId_t *instId /* OUTPUT */
)
```

Parameters

cellId	Specifies the ID of the cell that contains the instance (inst) or instance array (instArray).
instName	Specifies the name of the instance or instance array.
ignoreErrors	Specifies that error messages are not printed; returns FALSE.
instId	Returned object ID of the instance or instance array.

Return

MWXSuccess	Occurs when the function successfully gets the instance or instance array ID.
MWXFail	Occurs when the function fails to get the instance or instance array ID by name.

Dynamic Example

```
#include<stdio.h>
#include<stdlib.h>
#include "mwx.h"
#include "mwx_proto.h"

MWXSCM_t mwGetCellInstanceOrArrayByName (MWXSCM_t list)
{
    char progDescrString[50] = "Get CellInst/cellInstArray Id by name -";
    char instName[256];
    char cellName[256];
    MWXObjId_t instId;
    MWXCellId_t cellId;
    static int caseSensitive = 0;
    /*-----
    GET THE SCHEME LIST
    -----*/
    if (MWXSc_Scanf (list, "ss",cellName,instName) != 2) {
        MWX_Printf((char*)"Error.Synopsis : getCellInstanceOrArrayByName cellName
instName/instArrayName\n");
        return (MWX_FALSE_SCM);
    }
    /*-----
    HELLO
```

```

-----*/
MWX_Printf((char*)"\\n%s Testing...\\n", progDescrString);

if (MWXDb_Open_Cell (cellName, 'r', &cellId) != MWXSuccess) {
    MWX_Printf((char*) "Error : Can't open cell %s\\n",cellName);
    return (MWX_FALSE_SCM);
}

if (MWXDb_Get_CellInstanceOrArray_ByName(cellId,instName,0, &instId) !=
MWXSuccess) {
    MWX_Printf((char*) "Error : Can't get cellInst/cellInstArray %s by Name
\\n",instName);
    return(MWX_FALSE_SCM);
}

MWX_Printf((char*) "instId of inst/instArray %s is %d\\n",instName, instId);
/*-----*/
    DONE
-----*/
MWXDb_Close_Cell(cellId);
MWXDb_Close_Lib();
MWX_Printf((char*)" %s DONE TESTING! \\n\\n",progDescrString);
return (MWX_TRUE_SCM);
}
void reg_getCellInstanceOrArrayByName() {
    MWXSc_Register_Function((char*)"getCellInstanceOrArrayByName",
    mwGetCellInstanceOrArrayByName);
}

```

Stand-Alone Example

```

int GetCellInstanceOrArrayByName (
    char *cellName,
    char *instName
)
{
    MWXObjId_t instId;
    MWXCellId_t cellId;

    if (MWXDb_Open_Cell (cellName, 'r', &cellId) != MWXSuccess) {
        MWX_Printf((char*) "Error : Can't open cell %s\\n",cellName);
        return (MWXFail);
    }

    if (MWXDb_Get_CellInstanceOrArray_ByName(cellId,instName,0, &instId) !=
MWXSuccess) {
        MWX_Printf((char*) "Error : Can't get cellInst/cellInstArray %s by Name
\\n",instName);
        return(MWXFail);
    }

    MWX_Printf((char*) "instId of inst/instArray %s is %d\\n",instName, instId);
    /*-----*/
        DONE
    -----*/
    MWXDb_Close_Cell(cellId);
    MWXDb_Close_Lib();
    MWX_Printf((char*)" DONE TESTING! \\n\\n");
    return (MWXSuccess);
}

```

MWXDb_Get_CellInstanceOrArray_ByName

```
}
```

See Also

[MWXDb_Get_CellInstance_ByName](#)

MWXDb_Get_CellInstCellType_Bit

Sets or specifies the CellInst cellType bit.

Syntax

```
int MWXDb_Get_CellInstCellType_Bit(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t cellInstId, /* INPUT */  
    int32 *bitConst /* OUTPUT */  
)
```

Parameters

cellId	Database object ID of the cell to which the information applies.
cellInstId	Database object ID of the cellinst to which the information applies.
bitConst	Bit constant that applies to the cell. Valid values are MWXcMaskLayoutViewType MWXcNetListViewType MWXcSchematicViewType MWXcSymbolicViewType MWXcLogicalModelViewType

Return

MWXSuccess	Occurs if the CellType bit is successfully set.
MWXFail	Occurs if the CellType bit is not successfully set.

Example

```
if (MWXDb_Get_CellInstCellType_Bit(cellId,  
cellInstId,&bitConst)!=MWXSuccess){  
    MWX_Printf("Fail to set cellInst cell type bit\n");  
    return (MWXFail);  
}  
return (MWXSuccess);
```

MWXDb_Get_CellInstFromCellInstMaster

Retrieves the database identifier for the cell instance. The output parameters of the function are undefined if the function does not return `MWXSuccess`.

Syntax

```
int32
MWXDb_Get_CellInstFromCellInstMaster (
    MWXCellId_t cellId /* INPUT */
    MWXObjId_t cellInstMasterId /* INPUT */
    int32*objCount /* OUTPUT */
    MWXObjId_t**objectsIdArray/* OUTPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell containing the cell instance master.
<code>cellInstMasterId</code>	Specifies the database identifier of the cell instance master from which the <code>cellInst</code> is being retrieved.
<code>objCount</code>	Specifies the number of cell instances returned in <code>objectsIdArray</code> .
<code>objectsIdArray</code>	Specifies the array of the database identifiers for the cell instances retrieved.

Return

<code>MWXSuccess</code>	Occurs if the <code>cellInst</code> is successfully retrieved.
<code>MWXFail</code>	Occurs if the <code>cellInst</code> is not successfully retrieved.

Dynamic Example

```
rc = MWXDb_Get_CellInstFromCellInstMaster(cellId, cellInstId,
                                           &objectCount, &objectsIdArray);
switch (rc) {
    case MWXSuccess:
        for (i = 0; i < objectCount; i++){
            sprintf(buf, "(define objects (append objects '(%d)))",
                    objectsIdArray[i]);
            MWXSc_Evaluate (buf);
        }
        if (objectCount > 0) free (objectsIdArray);
        return (MWXSc_Make_IntegerSCM (objectCount));
    case MWXFail:
        MWX_Printf("ERROR -- Failed to get cell instance\n");
}
```

```

        return (MWXFail);
default:
    MWX_Printf("ERROR -- unknown return code \n");
    return (MWXFail);
}

```

MWXDb_Get_CellLatestVersion

Retrieves the latest version number of a given cell.

Syntax

```

public int
MWXDb_Get_CellLatestVersion(
    char *cellName, /* INPUT */
    char *viewName, /* INPUT */
    int32 *version, /* OUTPUT */
)

```

Parameters

cellName	Specifies the name of the cell to receive the latest version number.
viewName	Specifies the view name of the cell to receive the latest version number.
version	Specifies the latest version number of the specified cell.

Return

MWXNoLibraryOpened	Occurs when no library opened.
MWXSuccess	Occurs when the latest version of the cell was successfully retrieved.
MWXFail	Occurs when Milkyway fails to retrieve the latest version of the cell.

Example

```

int GetLatestCellVersion (
    char    *cellName,
    char    *viewName )
{
    int latestVersion;
    if (MWXDb_Get_CellLatestVersion(cellNames, viewNames,
        &latestVersion) != MWXSuccess) {

```

```

        MWX_Printf("FAIL - Unable to get the cell latest
                    version\n");
        exit (MWXFail);
    }
    return (latestVersion);
}

```

MWXDb_Get_CellName_ByCellId

Gets the full name of the cell (`cellName.viewName` and `version`) for the specified cell ID (`cellId`).

Syntax

```

int MWXDb_Get_CellName_ByCellId (
    MWXCellId_t cellId, /* INPUT */
    char **cellName, /* OUTPUT */
    char **viewName, /* OUTPUT */
    int *version, /* OUTPUT */
)

```

Parameters

<code>cellId</code>	Specifies the cell to use to query the full name.
<code>cellName</code>	Specifies the returned cell name of the specified cell.
<code>viewName</code>	Specifies the returned view name of the specified cell.
<code>version</code>	Specifies the returned version of the specified cell.

Return

<code>MWXSuccess</code>	Occurs if the command successfully retrieves the cell specified in <code>cellId</code> .
<code>MWXFail</code>	Occurs if the command fails to get <code>cellName</code> .

Dynamic Example

```

#include<stdio.h>
#include<stdlib.h>
#include "mwX.h"
#include "mwX_proto.h"

MWXSCM_t mwGetCellNameByCellId (MWXSCM_t list)
{
    char progDescrString[50] = "Get CellName By CellId -";
    char *cellName;
    MWXCellId_t cellId;

```

MWXDb_Get_CellName_ByCellId

```

char *viewName;
int version;
static int caseSensitive = 0;
/*-----
   GET THE SCHEME LIST
   -----*/
if (MWXSc_Scanf (list, "o",&cellId) != 1) {
    MWX_Printf((char*)"Error.Synopsis : getCellNameByCellId cellId \n");
    return (MWX_FALSE_SCM);
}
/*-----
   HELLO
   -----*/
MWX_Printf((char*)" \n%s Testing...\n", progDescrString);

/* Get cellName by cellId */

if (MWXDb_Get_CellName_ByCellId(cellId, &cellName, &viewName,&version)
    != MWXSuccess) {
    MWX_Printf((char*)"Error : Can't get cellName by cellId\n");
    return(MWX_FALSE_SCM);
}

MWX_Printf((char *) "CellName of cellId %d is %s.%s;%d\n", cellId, cellName,
           viewName,version);
/*-----
   DONE
   -----*/
MWX_Printf((char*)" %s DONE TESTING! \n\n",progDescrString);
return(MWX_TRUE_SCM);
}
void reg_getCellNameByCellId(){

MWXSc_Register_Function((char*)"getCellNameByCellId",mwGetCellNameByCellId);
}

```

Stand-Alone Example

```

int GetCellNameByCellId (MWXCellId_t cellId)
{
    char *cellName;
    char *viewName;
    int version;
    static int caseSensitive = 0;

    /* Get cellName by cellId */

    if (MWXDb_Get_CellName_ByCellId(cellId, &cellName, &viewName,&version)
        != MWXSuccess) {
        MWX_Printf("Error : Can't get cellName by cellId\n");
        return MWXFail;
    }

    MWX_Printf("CellName of cellId %d is %s.%s;%d\n", cellId, cellName,
              viewName,version);
    /*-----
       DONE
       -----*/
    MWX_Printf("%s DONE TESTING! \n");
}

```

```

return MWXSuccess;
}

```

See Also

[MWXDb_Get_CellFullName](#)
[MWXDb_Get_CellNames_ByView](#)

MWXDb_Get_CellNames_ByView

Returns a list of cells by view in the specified library.

Syntax

```

public int
MWXDb_Get_CellNames_ByView(
    MWXLibId_t libId /* INPUT */
    char *viewName /* INPUT */
    int32 **cellCount /* OUTPUT */
    MWXDBString_t **cellNames /* OUTPUT */
)

```

Parameters

libId	Specifies the library identification number.
viewName	Specifies the view name to receive the list of cells.
cellCount	Specifies the total retrieved cells by view name.
cellNames	Specifies the list of retrieved cells by the view name.

Return

MWXNoLibraryOpened	Occurs when no library opened.
MWXSuccess	Occurs when the list of cells by view is successfully retrieved.
MWXFail	Occurs when Milkyway fails to retrieve the list of cells by view.

Example

```

void Print_CellNames_ByView(
    MWXLibId_t libId,
    char *viewName
)
{

```

```

    int cellCount;
    MWXDBString_t *cellNames;

    if (MWXDb_Get_CellNames_ByView(libId, viewName,
        &cellCount, &cellNames) != MWXSuccess) {
        MWX_Printf("FAIL - Unable retrieved cell names by
            view name\n");
        exit (MWXFail);
    }
    MWX_Printf("List of cell Name with view %s are
        :\n",viewName);
    for (i =0; i < cellCount ; i++) {
        MWX_Printf("%s\n",cellNames[i]);
    }
    if (cellNames)
        free(cellNames);
    MWX_Printf("Done!\n");
}

```

MWXDb_Get_CellProperty_ByName

Retrieves a property attached to a cell. The output parameters of the function are undefined if the function does not return MWXSuccess.

Note:

There is no need to free memory for the propertyValue field.

Syntax

```

int MWXDb_Get_CellProperty_ByName (
    MWXCellId_t cellId, /* INPUT */
    char *propertyName, /* INPUT */
    MWXObjId_t *propertyId, /* OUTPUT */
    unsigned char *propertyDataType, /* OUTPUT */
    MWXPropValue_t *propertyValue, /* OUTPUT */
    uint16 *propertySize /* OUTPUT */
)

```

Parameters

cellId	Specifies a database identifier of the cell that owns the property.
propertyName	Specifies the name of the property to retrieve.
propertyId	Specifies the database identifier of the property retrieved.
propertyDataType	Specifies the type of data the property contains; valid values are defined in the mwX.h file and include MWXcPropBooleanType MWXcPropIntegerType MWXcPropFloatingType MWXcPropStringType MWXcPropDataStructType MWXcPropTimeType
propertyValue	Specifies the value of the property.
propertySize	Specifies the number of bytes the data requires for storage (in the case of a string, the size is the length of the string; in all other cases, it is the size of the data object).

Return

MWXSuccess	Occurs if the property is successfully retrieved.
MWXFail	Occurs if Milkyway fails to retrieve the property.

Example

```
if (MWXDb_Get_CellProperty_ByName (cellId, propertyName,
    &propertyId,&propertyDataType, &propertyValue, &propertySize) == MWXSuccess) {
    switch (propertyDataType) {
        case MWXcPropBooleanType:
            ...
            break;
        case MWXcPropIntegerType:
            ...
            break;
        case MWXcPropFloatingType:
            ...
            break;
        case MWXcPropStringType:
            ...
            break;
        case MWXcPropDataStructType:
            ...
            break;
        case MWXcPropTimeType:
            MWX_Printf("INFO -- found time property.\n");
            break;
        default:
            MWX_Printf("ERROR -- Unknown property type returned.\n");
    }
```

MWXDb_Get_CellProperty_ByName

```

    }
}

```

MWXDb_Get_CellType_Bit

Gets or specifies the type of a cell.

Syntax

```

int MWXDb_Get_CellType (
    MWXCellId_t cellId, /* INPUT */
    int32 *bitConst /* OUTPUT */
)

```

Parameters

cellId	Database object ID of the cell to which the information applies.
bitConst	Bit constant that applies to the cell. Valid values: MWXcMaskLayoutViewType MWXcNetListViewType MWXcSchematicViewType MWXcSymbolicViewType MWXcLogicalModelViewType

Return

MWXSuccess	Occurs if the cellTypebit constant is successfully retrieved.
MWXFail	Occurs if the cellTypebit constant is not successfully retrieved.

Example

```

int getCellTypeBit (MWXCellId_t cellId) {
    int32 bitConst;
    int32 rc;
    rc = MWXDb_Get_CellType_Bit (cellId, bitConst);
    if (rc == MWXFail) {
        MWX_Printf("Fail to get cellType bit\n");
        return (MWXFail);
    }
    switch (bitConst) {
    case MWXcMaskLayoutViewType:
        MWX_Printf("This is mask layout view cell type);
        break;
    case MWXcNetListViewType:
        MWX_Printf("This is netList view cell type);
        break;
    case MWXcSchematicViewType:
        MWX_Printf("This is Schematic view cell type);
        break;
    }
}

```



```

case MWXcSymbolicViewType:
    MWX_Printf("This is Symbolic view cell type);
    break;
case MWXcLogicalModelViewType:
    MWX_Printf("This is Schematic view cell type);
    break;
default:
    MWX_Printf("*** UNKNOWN CELL TYPE);
    return(MWXFail);
}
return (MWXSuccess);
}

```

MWXDb_Get_ColorIdByName

Retrieves the color ID associated with the given color from the technology file.

Syntax

```

int32 MWXDb_Get_ColorIdByName (
    char *colorName, /* INPUT */
    int32 *colorId /* OUTPUT */
)

```

Parameters

colorName	Specifies the name of the color from the technology file.
colorId	Returns the ID associated with the given color.

Return

MWXSuccess	Occurs if the color ID is successfully retrieved.
MWXNoLibraryOpened	Occurs if the design library is not open.
MWXFail	Occurs if the color ID is not retrieved for any other reason.

Dynamic Example

```

MWXSCM_t mwGetColorIdByName(MWXSCM_t list) {
    char colorName[256];
    int colorId;
    int rc;

    if (MWXSc_Scanf(list, "s", colorName) != 1) {
        MWX_Printf("ERROR - Usage: mwGetColorIdByName colorName\n");
        return MWX_FALSE_SCM;
    }
}

```

```

rc = MWXDb_Get_ColorIdByName(colorName, &colorId);
switch ( rc ) {
case MWXSuccess:
    MWX_Printf("Color %s has ID: %d\n", colorName, colorId);
    return MWX_TRUE_SCM;
case MWXNoLibraryOpened:
    MWX_Printf("ERROR -- No library openend. MWXDb_Get_ColorIdByName "
               "failed\n", rc);
    return MWX_FALSE_SCM;
case MWXFail:
    MWX_Printf("Failed to retrieve colorId.\n");
    return MWX_FALSE_SCM;
default:
    MWX_Printf("ERROR -- Unknown return code %d. MWXDb_Get_ColorIdByName "
               "failed\n", rc);
    return MWX_FALSE_SCM;
}
}

```

Stand-Alone Example

```

int GetColorIdByName(char *colorName, int *colorId) {

    int rc;

    MWX_Printf("INFO -- MWXDb_Get_ColorIdByName.\n");
    rc = MWXDb_Get_ColorIdByName (colorName, colorId);

    switch (rc) {
case MWXSuccess:
    MWX_Printf("Color %s has ID: %d\n", colorName, *colorId);
    return MWXSuccess;
case MWXNoLibraryOpened:
    MWX_Printf("ERROR -- No library openend. MWXDb_Get_ColorIdByName "
               "failed\n", rc);
    return MWXFail;
case MWXFail:
    MWX_Printf("Failed to retrieve colorId.\n");
    return MWXFail;
default:
    MWX_Printf("ERROR -- Unknown return code %d. MWXDb_Get_ColorIdByName "
               "failed\n", rc);
    return MWXFail;
}
}

```

See Also

MWXDb_Get_ColorName

MWXDb_Get_ConnectedObject_ById

This function retrieves connected objects by object ID.

Syntax

```
int MWXDb_Get_ConnectedObject_ById(  
MWXCellId_t cellId /* INPUT */  
MWXObjId_t objId, /* INPUT */  
MWXObjId_t **objectsIdArray /* OUTPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell containing objects to retrieve
objId	Specifies the database identifier of the object that you want to get the connected object info from
objectsIdArray	Specifies the array of database identifiers for connected objects retrieved

Return

MWXSuccess	Occurs if successfully retrieved objects.
MWXFail	Occurs if fail to retrieve objects

Example

```
MWXObjId_t* getConnectedObject (  
    MWXCellId_t cellId,  
    MWXObjId_t objectId)  
{  
    int32 rc;  
    MWXObjId_t * objIdArray;  
    rc = (MWXDb_Get_ConnectedObject_ById(cellId, objectId, &objIdArray);  
    if (rc == MWXFail) {  
        MWX_Printf("Fail to get connected object to input object \n");  
        return 0;  
    }  
    return (objIdArray);  
}
```

MWXDb_Get_Constants

Gets the value of the requested constant. Constant values are typically used to mask bitwise database values.

Syntax

```
int MWXDb_Get_Constants(  
    char *constantName, /* INPUT */  
    int32 *constantInt /* OUTPUT */  
)
```

Parameters

constantName Specifies the name of the constant to retrieve.

constantInt Specifies the value of the requested constant.

Return

MWXSuccess Occurs if the constant is successfully retrieved.

MWXFail Occurs if the constant is not successfully retrieved.

Example

```
rc = MWXDb_Get_Constants ("dbbStandardCell",  
    &constantInt);  
switch (rc) {  
    case MWXSuccess:  
        break;  
    case MWXFail:  
        MWX_Printf("ERROR -- failed to get constant\n");  
        break;  
}
```

Scheme Equivalent

CONSTANT

(see Physical Implementation Online Help)

MWXDb_Get_ContactCode_ByName

Gets contact code information from the specified library ID and contact code name.

Syntax

```
public int  
MWXDb_Get_ContactCode_ByName (  
    MWXLibId_t                      libId,                      /* INPUT */  
    char                            *contactCodeName,          /* INPUT */  
    MWXContactCodeTech_t          *contactCodeInfo          /* OUTPUT */  
)
```

Parameters

libId	Specifies the database identifier of the library from which to retrieve the contact code information.
contactCodeName	Specifies the contact code name with which to get the corresponding contact code information.
contactCodeInfo	Specifies the contact code information.

Return

MWXSuccess	Occurs if the command successfully retrieves the contact code information.
MWXFail	Occurs if the command fails to retrieve the contact code information.

Example

```
int32 getContactCodeInfo (
    MWXLibId_t libId,
    char *contactCodeName
) {
    MWXContactCodeTech_t contactCodeInfo;
    rc = MWXDb_Get_ContactCode_ByName(libId, contactCodeName, &contactCodeInfo);
    if (rc == MWXSuccess){
        MWX_Printf("contact code information : \n");
        MWX_Printf("contact code number: %d\n", contactCodeInfo.contactCodeNumber);
        MWX_Printf("cutLayer: %s\n", contactCodeInfo.cutLayer);
        MWX_Printf("lowerLayer: %s\n", contactCodeInfo.lowerLayer);
        MWX_Printf("upperLayer: %s\n", contactCodeInfo.upperLayer);
        MWX_Printf("isDefaultContact: %d\n", contactCodeInfo.isDefaultContact);
        MWX_Printf("isFatContact: %d\n", contactCodeInfo.isFatContact);
        MWX_Printf("cutWidth: %d\n", contactCodeInfo.cutWidth);
        MWX_Printf("cutHeight: %d\n", contactCodeInfo.cutHeight);
        MWX_Printf("lowerLayerEncWidth: %d\n", contactCodeInfo.lowerLayerEncWidth);
        MWX_Printf("lowerLayerEncHeight: %d\n",
            contactCodeInfo.lowerLayerEncHeight);
        MWX_Printf("upperLayerEncWidth: %d\n", contactCodeInfo.upperLayerEncWidth);
        MWX_Printf("upperLayerEncHeight: %d\n",
            contactCodeInfo.upperLayerEncHeight);
        MWX_Printf("minCutSpacing: %d\n", contactCodeInfo.minCutSpacing);
        MWX_Printf("maxNumRowsNonTurning: %d\n",
            contactCodeInfo.maxNumRowsNonTurning);
        MWX_Printf("unitMinCapacitance: %d\n", contactCodeInfo.unitMinCapacitance);
        MWX_Printf("unitNomCapacitance: %d\n", contactCodeInfo.unitNomCapacitance);
        MWX_Printf("unitMaxCapacitance: %d\n", contactCodeInfo.unitMaxCapacitance);
        MWX_Printf("unitMinResistance: %d\n", contactCodeInfo.unitMinResistance);
        MWX_Printf("unitNomResistance: %d\n", contactCodeInfo.unitNomResistance);
        MWX_Printf("unitMaxResistance: %d\n", contactCodeInfo.unitMaxResistance);
        MWX_Printf("unitMinInductance: %d\n", contactCodeInfo.unitMinInductance);
        MWX_Printf("unitNomInductance: %d\n", contactCodeInfo.unitNomInductance);
        MWX_Printf("unitMaxInductance: %d\n", contactCodeInfo.unitMaxInductance);
    }
```

MWXDb_Get_ContactCode_ByName

```

MWX_Printf("temperatureCoeff: %f\n", contactCodeInfo.temperatureCoeff);
}
else{
    MWX_Printf("failed to get contact code information\n");
    return (MWXFail);
}

return (MWXSuccess);

```

MWXDb_Get_ContactCodeIdx_ByName

Gets the contact code index from the given name.

Syntax

```

int MWXDb_Get_ContactCodeIdx_ByName (
    MWXLibId_t libId, /* INPUT */
    char *contactCodeName, /* INPUT */
    uchar *contactCodeIdx /* OUTPUT */
)

```

Parameters

libId	Specifies the database identifier of the open library.
contactCodeName	Specifies the contact code name defined in the technology file.
contactCodeIdx	The contact code index.

Return

MWXSuccess	Occurs if successfully retrieved contact code index.
MWXFail	Occurs if fail to retrieve contact code index.
MWXNoLibraryOpened	Occurs if there is no library currently open.

Example

```

int getContactCodeIdx (
    MWXLibId_t libId,
    char *contactCodeName,
    uchar *contactCodeIdx
) {
    if (MWXDb_Get_ContactCodeIdx_ByName(libId, contactCodeName &contactCodeIdx)
    == MWXFail) {
        MWX_Printf("Fail to get contact code index by name\n");
        return (MWXFail);
    }
    return (MWXSuccess);
}

```

MWXDb_Get_ContactCodeNames

Get contact code numbers and corresponding contact code names from the current opened library. This function will malloc memory locations for contactCodeNumbers and contactCodeNames.

It is the users' responsibility to free the two memory locations, for example, `free(contactCodeNumbers)` or `free(contactCodeNames)`.

Syntax

```
int MWXDb_Get_ContactCodeNames(
    MWXLM_Id_t libId,          /* INPUT */
    int *count,                /* OUTPUT */
    int **contactCodeNumbers   /* OUTPUT */
    MWX_tfContactName_t **contactCodeNames /* OUTPUT */
)
```

Parameters

libId	Database identifier for the Milkyway library
count	Returned total count of contact code names
contactCodeNumbers	Returned array of contact code numbers
contactCodeNames	Returned array of corresponding contact code names

Return

MWXSuccess	Successfully gets contact code information
MWXFail	Fail to get contact code information

Example

```
int contactCodeCount;
int* contactCodeNumbers;
MWX_tfContactName_t *contactCodeNames;
int32 viaLayerNumber, lowerLayerNumber, upperLayerNumber;
int i, rc;

rc = MWXDb_Get_ContactCodeNames(myLibId, &contactCodeCount,
&contactCodeNumbers, &contactCodeNames);
if ( rc == MWXFail) {
    MWX_Printf("ERROR -- failed to get contactCodes' info\n");
    exit (MWXFail);
} else {
    for (i=0; i<contactCodeCount; i++) {
        MWXDb_Get_TechContactLayer(*(contactCodeNumbers+i), &
viaLayerNumber, &lowerLayerNumber, &upperLayerNumber);
        MWX_Printf("ContactCode number : %d\n",
```

```

                                *(contactCodeNumber+i));
MWX_Printf("ContactCode Name : %s \n",
          *(contactCodeNames+i));
MWX_Printf("viaLayerNumber : %d \n",viaLayerNumber);
MWX_Printf("lowerLayerNumber : %d\n",lowerLayerNumber);
MWX_Printf("upperLayerNumber : %d\n",upperLayerNumber);
}
}
free(contactCodeNumbers);
free(contactCodeNames);

```

MWXDb_Get_ContactGeom

Retrieves all contact geometry, such as netId, layer, left, bottom, right, and top, that applies to three layers of the contact.

Syntax

```

int MWXDb_Get_ContactGeom (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t contactId, /* INPUT */
    MWXObjId_t netId[], /* OUTPUT */
    int32 layer[], /* OUTPUT */
    int32 left[], /* OUTPUT */
    int32 bottom[], /* OUTPUT */
    int32 right[], /* OUTPUT */
    int32 top[] /* OUTPUT */
)

```


Parameters

cellId	Specifies the database identifier of the cell containing the contact.
contactId	Specifies the database identifier of the contact object.
netId	Specifies the database identifier of the net connected to three layers of the contact.
layer	Three layer numbers of the contact.
left	The three left coordinates of the bbox, in DB units.
bottom	The three bottom coordinates of the bbox, in DB units.
right	The three right coordinates of the bbox, in DB units.
top	The three top coordinates of the bbox, in DB units.

Return

MWXSucces	Occurs if all the contact geometry is successfully retrieved.
MWXFail	Occurs if all the contact geometry is not successfully retrieved.

Example

```
int32 printContactGeom (
    MWXCellId_t cellId,
    MWXObjId_t contactId)
{
    MWXObjId_t netId[3];
    char *net[3];
    int32 layer[3];
    int32 left[3];
    int32 bottom[3];
    int32 right[3];
    int32 top[3];
    int32 i, rc;

    rc =
MWXDb_Get_ContactGeom(cellId,contactId,netId,layer,left,bottom,right,top);
    if (rc == MWXFail) {
        MWX_Printf("Fail to get contactGeom \n");
        return (MWXFail);
    }
    else {
        MWX_Printf("Contact geometry Information : \n");
        for (i=0;i<3;i++) {
            rc = MWXDb_Get_Net_name(cellId,netId[i],&netName[i]);
            if (rc == MWXFail) {
                MWX_Printf("Fail to get Net name\n");
            }
        }
    }
}
```

MWXDb_Get_ContactGeom

```

        exit (MWXFail);
    }
    MWX_Printf ("%t Net Name : %s\n",netName);
    MWX_Printf ("%t layer      : %d\n", layer[i];
    MWX_Printf ("%t bbox        : (%s, %s, %s, %s) \n", left, bottom, right, top);
    }
}
return (MWXSuccess);
}

```

Scheme Equivalent

dbFetchContactGeom

(see Physical Implementation Online Help)

MWXDb_Get_ContactName_ByIndex

Gets the contact name from the given index.

Syntax

```

int MWXDb_Get_ContactName_ByIndex (
    MWXLibId_t libId, /* INPUT */
    int32 index, /* INPUT */
    char **contactName /* OUTPUT */
)

```

Parameters

libId	Specifies the database identifier of the library that was opened.
index	Specifies the index defined in the technology file.
contactName	Specifies the returned contact name.

Return

MWXSuccess	Occurs if the contact name from the given index is successfully retrieved.
MWXFail	Occurs if the contact name from the given index is not successfully retrieved.

Example

```
char *contactName;
```

```
rc = MWXDb_Get_ContactName_ByIndex(libId, index, &contactName);
```

```

switch (rc) {
case MWXSuccess:
    MWX_Printf("Contact Name is (%s)\n", contactName);
    return (MWXSuccess);
case MWXFail:
    MWX_Printf("ERROR -- Failed to operate.\n");
    return (MWXFail);
default:
    MWX_Printf("ERROR -- unknown return code from
MWXDb_Get_ContactName_ByIndex\n");
    return (MWXFail);
}

```

MWXDb_Get_CurrentLibId

Retrieves the database identifier of the currently open design library.

Syntax

```

int MWXDb_Get_CurrentLibId (
    MWXLibId_t *libraryId /* OUTPUT */
)

```

Parameters

libraryId	Returns the database identifier of the design library.

Return

MWXSuccess	Occurs if the library database identifier is successfully retrieved.
MWXNoLibraryOpened	Occurs if the design library is not currently open.

Dynamic Example

```

MWXSCM_t mwGetCurrentLibId(MWXSCM_t list) {

    int rc;
    MWXLibId_t libraryId;

    MWX_Printf("INFO -- MWXDb_Get_CurrentLibId.\n");
    rc = MWXDb_Get_CurrentLibId(&libraryId);

    switch (rc) {
case MWXSuccess:
    MWX_Printf("Successfully retrieved library Id. Library Id = %d.\n",
        libraryId);
    return (MWXSc_Make_DBIIdSCM(libraryId));
case MWXNoLibraryOpened:
    MWX_Printf("ERROR -- No library opened. MWXDb_Get_CurrentLibId

```

```

        failed.\n");
    return (MWX_FALSE_SCM);
default:
    MWX_Printf("UNKNOWN ERROR %d from MWXDb_Get_CurrentLibId.\n", rc);
    return (MWX_FALSE_SCM);
}
}

```

Stand-Alone Example

```

int GetCurrentLibId(MWXLibId_t *libraryId) {

    int rc;

    MWX_Printf("INFO -- MWXDb_Get_CurrentLibId.\n");
    rc = MWXDb_Get_CurrentLibId(libraryId);

    switch (rc) {
    case MWXSuccess:
        MWX_Printf("Successfully retrieved library Id.  Library Id = %d.\n",
            *libraryId);
        return (MWXSuccess);
    case MWXNoLibraryOpened:
        MWX_Printf("ERROR -- No library opened.  MWXDb_Get_CurrentLibId
            failed.\n");
        return (MWXFail);
    default:
        MWX_Printf("UNKNOWN ERROR %d from MWXDb_Get_CurrentLibId.\n", rc);
        return (MWXFail);
    }
}

```

Scheme Equivalent

dbGetCurrentLibId
(see Physical Implementation Online Help)

See Also

[MWXDb_Create_Lib](#)
[MWXDb_Open_Lib](#)
[MWXDb_Open_Library_ByAccessMode](#)

MWXDb_Get_DBUPerUDistU

Gets the database unit per user unit distance.

Syntax

```

int MWXDb_Get_DBUPerUDistU (
    MWXLibId_t LibId, /* INPUT */
    int32 *dbuPerUDistU /* OUTPUT */
)

```

Parameters

LibId	Specifies the ID of the library.
dbuPerUDistU	The database unit per user distance unit.

Return

MWXSuccess	Occurs if the database unit per user distance is successfully retrieved.
MWXFail	Occurs if the database unit per user distance is not successfully retrieved.

Example

```
rc = MWXDb_Get_DBUPerUDistU (libId, &dbuPerUDistU);
if (rc != MWXSuccess)
    MWX_Printf("Fail to get DBUPerUDistU\n");
    return 0;
else
    return(dbuPerUDistU);
```

MWXDb_Get_DefaultContactByMetalLayers

Get the default contact from two given layers.

Syntax

```
int MWXDb_Get_DefaultContactByMetalLayers (
    int32 layer1, /* INPUT */
    int32 layer2, /* INPUT */
    uchar *contactCodeIdx /* OUPUT */
)
```

Parameters

layer1	Specifies the first metal layer from which you want to get the default contact.
layer2	Specifies the second metal layer from which you want to get the default contact.
contactCodeIdx	The contactCode index.

Return

MWXSuccess	Occurs if the default contact is successfully retrieved.
MWXFail	Occurs if the default contact is not successfully retrieved.

Example

```
rc = MWXDb_Get_DefaultContactByMetalLayers(layer1, layer2, &contactCodeIdx);
switch (rc) {
    case MWXSuccess:
        MWX_Printf("The contactCodeIdx is %d\n",contactCodeIdx);
        return (contactCodeIdx);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to get default contact by layerNumber\n");
        return (0);
    default:
        MWX_Printf("ERROR -- unknown return code from
MWXDb_Get_DefaultContactByMetalLayers\n");
        return (0);
}
```

MWXDb_Get_DefaultContactByViaLayer

Gets the default contact from given vialayer.

Syntax

```
int MWXDb_Get_DefaultContactByViaLayer (
    int32 viaLayer, /* INPUT */
    uchar *contactCodeIdx /* OUPUT */
)
```

Parameters

<code>viaLayer</code>	Specifies the first via layer from which you want to get the default contact.
<code>contactCodeIdx</code>	The <code>contactCode</code> index.

Return

<code>MWXSucccess</code>	Occurs if the default contact is successfully retrieved.
<code>MWXFail</code>	Occurs if the default contact cannot be successfully retrieved.

Example

```
rc = MWXDb_Get_DefaultContactByViaLayer(viaLayer, &contactCodeIdx);
switch (rc) {
    case MWXSucccess:
        MWX_Printf(" The contactCodeIdx of via %d is %d\n",viaLayer,
            contactCodeIdx);
        return (contactCodeIdx);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to get default contact by layerNumber\n");
        return (0);
    default:
        MWX_Printf("ERROR -- unknown return code from
MWXDb_Get_DefaultContactByViaLayer\n");
        return (0);
}
```

MWXDb_Get_EditCell

Gets the current cellId.

Syntax

```
int MWXDb_Get_EditCell (
    MWXCellId_t *editCellId    /* OUTPUT */
)
```

Parameters

`editCellId` The ID of the current open cell.

Return

`MWXSuccess` Occurs if the `editCellId` is successfully retrieved.

`MWXFail` Occurs if the `editCellId` is not successfully retrieved.

Example

```
if(MWXDb_Get_EditCell(&cellId) != MWXSuccess) {  
    MWX_Printf("Fail to Get edit cell \n");  
    return (MWX_FALSE_SCM);  
}  
  
    return (MWXSc_Make_DBIIdSCM(cellId));
```

Scheme Equivalent

`geGetEditCell`

(see Physical Implementation Online Help)

MWXDb_Get_FatContactByMaskName

Gets the fat contact from given mask name.

Note:

There is no need to free memory for the `ContactCodeIdx` field.

Syntax

```
int MWXDb_Get_FatContactByMaskName (  
    char *maskName, /* INPUT */  
    int32 *numberOfcontactCodeIdx, /* OUTPUT */  
    uchar **contactCodeIdx /* OUPUT */  
)
```


Parameters

maskName	Specifies the mask name of the fat contact that should get the data back.
numberOfcontactCodeIdx	Number of the contact code index.
contactCodeIdx	The contact code index.

Return

MWXSuccess	Occurs if the fat contact is successfully retrieved.
MWXFail	Occurs if the fat contact is not successfully retrieved.

Example

```
rc = MWXDb_Get_FatContactByMaskName (maskName, &numberOfcontactCodeIdx,
&contactCodeIdx);
switch (rc) {
    case MWXSuccess:
        MWX_Printf("Number of ContactCode:
%d\n", numberOfContactCodeIdx);
        MWX_Printf("ContactCodeIdx is %d
\n",contactCodeIdx);
        return (MWXSuccess);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to get fat contact by maskName\n");
        return (MWXFail);
    default:
        MWX_Printf("ERROR -- unknown return code from
MWXDb_Get_FatContactByMaskName\n");
        return (MWXFail);
}
```

MWXDb_Get_Group_ByName

Retrieves the group ID for a specific group object within a cell. The group is specified by groupName and groupSignature for the cell to which it belongs.

Syntax

```
#include "mwx.h"
# include "mwx_proto.h"
int MWXDb_Get_Group_ByName (
    MWXCellId_t cellId, /* INPUT */
    int32 groupSignature, /* INPUT */
    char *groupName, /* INPUT */
```

MWXDb_Get_Group_ByName

```

        MWXObjId_t *groupId /* OUTPUT */
    )

```

Parameters

<code>cellId</code>	Specifies the database identifier of cell containing the group object.
<code>groupSignature</code>	Combines with <code>groupName</code> to specify the group ID you want to retrieve from the database. Valid values: 0–255. A Synopsys application reserves group numbers 0–200 and 255 for predefined purposes.
<code>groupName</code>	Name of the group you want to get from the database. Valid values: Any string with 31 characters or fewer
<code>groupId</code>	Database identifier for the group you want.

Return

<code>MWXFail</code>	Occurs if the group ID is not successfully obtained.
<code>MWXNoLicense</code>	Occurs if the Synopsys license is not present.
<code>MWXSuccess</code>	Occurs if the group ID is successfully obtained.

Example

```

rc = MWXDb_Get_Group_ByName (cellId, groupnumber,
    groupName, &groupId);
switch (rc) {
    case MWXSuccess:
        return(groupId);
    case MWXFail:
        DMWX_Printf("ERROR -- Failed to get groupId \n");
        return (MWXFail);
    default:
        MWX_Printf("ERROR -- unknown return code from
            MWXDb_Get_Group_ByName\n");
        return (MWXFail);
}

```

MWXDb_Get_GroupMember

Gets an array of database identifiers for member cells of the specified group. It is your responsibility to recover the memory allocated for the `objectsIdArray` field when the data is no longer needed.

Syntax

```
int MWXDb_Get_GroupMember (
    MWXCellId_t cellId, /*INPUT*/
    MWXObjId_t groupId, /*INPUT*/
    int *objectCount, /*OUTPUT*/
    MWXObjId_t **objectsIdArray /*OUTPUT*/
)
```

Parameters

<code>cellId</code>	Database identifier of the cell that contains the group.
<code>groupId</code>	Database identifier of the group.
<code>objectCount</code>	Number of objects attached to this group.
<code>objectsIdArray</code>	An array of object identifiers for cells in this group. It is your responsibility to free the memory—for example, <code>free (*memberIdArray)</code> .

Return

<code>MWXFail</code>	Occurs if object identifiers are not successfully retrieved.
<code>MWXSuccess</code>	Occurs if object identifiers are successfully retrieved.

Example

```
rc=MWXDb_Get_GroupMember(cellId,groupId,&objCount, &objIdArray);
switch (rc) {
    case MWXSuccess:
        if(objCount>0) free(objIdArray);
        return (MWXSc_Make_IntegerSCM (objCount));
    case MWXFail:
        return (MWXFail);
}
```

MWXdB_Get_HierRefLibMode

This function gets the hierarchy reference library mode. The initial value is FALSE (0). When the same cell name exists in two or more reference libraries, the user needs to set the mode to TRUE before opening a child cell.

Syntax

```
int MWXdB_Get_HierRefLibMode(  
int *mode /* OUTPUT */  
)
```

Parameters

mode	initial is 0. The value is 1 when same cell name exists in two or more reference libraries.

Return

MWXSuccess	Occurs if successfully gets the hierRefLibMode.
MWXFail	Occurs if fail to get the hierRefLibMode

Example

```
rc = MWXdB_Get_HierRefLibMode(&mode);  
switch (rc) {  
    case MWXSuccess:  
        printf("The hierRefLibMode is %d\n",mode);  
        break;  
    case MWXFail:  
        printf("Fail to retrieve the hierRefLibMode\n");  
        break;  
    Default:  
        printf("Unknown return code \n");  
        break;  
}
```

MWXdB_Get_InputData

Gets the input data from the mouse click. The input data can be a point, a bbox, or a line.

Syntax

```
int MWXdB_Get_InputData (  
    char *shape  
)
```

Parameters

`shape` The shape of the input data from the mouse click. It can be `Point` or `point`, `BBox` or `bbox`, `Line` or `line`.

Return

`MWXSuccess` Occurs if the input data for the desired shape is successfully retrieved.

`MWXFail` Occurs if the input data is not successfully retrieved.

Note:

To get the input data from the mouse click, you need to have a callback function to control whether the data is already input or not.

Scheme Equivalent

`geOutputPoint`

(see Physical Implementation Online Help)

MWXDb_Get_Instances_ByBBoxSearch

Retrieves all cell instances that overlap the specified `bbox`. It is your responsibility to recover the allocated memory for the `objectsIdArray` field when the data is no longer needed.

Note:

If you attempt to make a sequence of this function call, call [MWXDb_Startup_ShapeStruct](#) first.

Syntax

```
int MWXDb_Get_Instances_ByBBoxSearch(  
    MWXCellId_t cellId, /* INPUT */  
    int32 left, /* INPUT */  
    int32 bottom, /* INPUT */  
    int32 right, /* INPUT */  
    int32 top, /* INPUT */  
    int32 *objectCount, /* OUTPUT */  
    MWXObjId_t **objectsIdArray /*OUTPUT */  
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell that was opened.
<code>left</code>	Specifies the x-coordinate for the left side of the bounding box.
<code>bottom</code>	Specifies the y-coordinate for the bottom of the bounding box.
<code>right</code>	Specifies the x-coordinate for the right side of the bounding box.
<code>top</code>	Specifies the y-coordinate for the top of the bounding box.
<code>objectCount</code>	The number of objects in the input.
<code>objectsIdArray</code>	An array of objects IDs. It is your responsibility to free the memory—for example, <code>free(*objectsIdArray)</code> .

Return

<code>MWXSuccess</code>	Occurs if the appropriate cell instances are successfully retrieved.
<code>MWXFail</code>	Occurs if the appropriate cell instances are not successfully retrieved.

Example

```
if (MWXDb_Get_Instances_ByBBoxSearch (cellId, left, bottom, right, top,
    &objCount, &objsIdArray) != MWXSuccess) {
    MWX_Printf("Can't get objects by bbox, invalid data\n");
    return (MWXFail);
}
if (objCount > 0) free (objsIdArray);
return (MWXSc_Make_IntegerSCM(objCount));
}
```

See Also

[MWXDb_Cleanup_ShapeStruct](#)
[MWXDb_Startup_ShapeStruct](#)

MWXDb_Get_LayerId_ByName

Retrieves the index of a layer. The index refers to the position of a layer in the layer table kept in memory, and the index is used by numerous commands to refer to the layer.

Syntax

```
int MWXDb_Get_LayerId_ByName (  
    char *layerName, /* INPUT */  
    int32 *layerId /* OUTPUT */  
)
```

Parameters

layerName	Specifies the name of the layer from which the index is retrieved.
layerId	Specifies the index value for the named layer.

Return

MWXSuccess	Occurs if the layer index is successfully retrieved.
MWXNoLibraryOpened	Occurs if no library is currently open.
MWXFail	Occurs if the layer index is not retrieved for any other reason.

Example

```
if (MWXDb_Get_LayerId_ByName (layerName, &layerId) != MWXSuccess) {  
    return (MWXSc_Make_IntegerSCM (layerId));  
} else {  
    return (MWXFail);  
}
```

Scheme Equivalent

dbFetchLayerIdByName
(see Physical Implementation Online Help)

MWXDb_Get_LayerId_ByNumber

Retrieves the index of a layer. The index refers to the position of a layer in the layer table kept in memory. The index is used by numerous commands to refer to the layer.

Syntax

```
int MWXDb_Get_LayerId_ByNumber (  
    int32 layerNumber, /* INPUT */  
    int32 *layerId /* OUTPUT */  
)
```

Parameters

<code>layerNumber</code>	Specifies the number of the layer for which the index is retrieved.
<code>layerId</code>	Specifies the index value for the named layer.

Return

<code>MWXSucces</code>	Occurs if the layer index is successfully retrieved.
<code>MWXNoLibraryOpened</code>	Occurs if no library is currently open.
<code>MWXFail</code>	Occurs if the layer index is not successfully retrieved for any other reason.

Example

```
if (MWXDb_Get_LayerId_ByName (layerNumber, &layerId) != MWXSucces) {  
    return (MWXSc_Make_IntegerSCM (layerId));  
} else {  
    return (MWXFail);  
}
```

Scheme Equivalent

`dbFetchLayerIdByNumber`
(see Physical Implementation Online Help)

MWXDb_Get_LayerName

Retrieves the name of a layer.

Note:

There is no need to free memory for the `layerName` field.

Syntax

```
int MWXDb_Get_LayerName (  
    int32 layerNumber, /* INPUT */  
    char **layerName /* OUTPUT */  
)
```


Parameters

layerNumber	Specifies the number of the layer for which the layer name is retrieved.
layerName	Specifies the name of the layer retrieved.

Return

MWXFail	Occurs if the layer name cannot be successfully retrieved from the layer number.
MWXSuccess	Occurs if the layer name is successfully retrieved.

Example

```
if (( rc = MWXDb_Get_LayerName(layerNumber, &layerName)) != MWXSuccess){
    MWX_Printf("Cannot get layerName for layer %d\n", layerNo);
    return (MWXFail);
}

return (MWXSuccess);
```

MWXDb_Get_LayerNames

Gets the layer numbers and corresponding layer names with the specified library identity. You must free the memory that is allocated for `layerNumbers` and `layerNames` when the data is no longer needed.

Syntax

```
int MWXDb_Get_LayerNames (
    MWXLibId_t libId, /* INPUT */
    int *count, /* OUTPUT */
    int **layerNumbers, /* OUTPUT */
    MWX_tfName_t **layerNames /* OUTPUT */
)
```

Parameters

libId	Specifies the database identifier of the library from which to get layer numbers and names.
count	Specifies the number of layer names found in the library.
layerNumbers	Specifies an array of layer numbers of each layer name.
layerNames	Specifies an array of layer names.

Return

MWXSuccess	Occurs if the layer names were successfully retrieved.
MWXFail	Occurs if the layer names were not successfully retrieved.

Example

```
int rc;
int *layerNumbers;
MWX_tfName_t **layerNames;

rc = MWXDb_Get_LayerNames (libId, &count, &layerNumbers, &layerNames);

switch ( rc ) {
    case MWXSuccess:
        MWX_Printf("there are %d layer names in this library.\n", count);
        for ( i = 0 ; i < count ; i ++ )
            MWX_Printf("    layer number : %d, layer name : %s \n",
                *(layerNumbers + i), layerNames + i);

        if ( layerNumbers )
            free ( layerNumbers );
        if ( layerNames )
            free ( layerNames );
        break;
    case MWXFail:
        MWX_Printf("failed to get layer names.\n");
        return 0;
}

return 1;
```

See Also

[MWXDb_Get_ContactCodeNames](#)

MWXDb_Get_LibAttachedFiles

Returns a list of attached files of the specified library.

Syntax

```
public int
MWXDb_Get_LibAttachedFiles(
    MWXLibId_t libId /* INPUT */
    int *attachedFileCnt /* OUTPUT */
    MWXDBString_t **attachedFiles /* OUTPUT */
    MWXDBString_t **pseudoAccessNames /* OUTPUT */
    int32 **fileExistStatus /* OUTPUT */
)
```

Parameters

libId	Specifies the design library database identification.
attachedFileCnt	Specifies the returned value for the total number of attached files in the library.
attachedFiles	Specifies the name list of all attached files in the library.
pseudoAccessNames	Specifies the name list of all pseudo access names in the library.
fileExistStatus	Specifies the list of file existence flag.

Return

MWXNoLibraryOpened	Occurs when no library opened.
MWXSuccess	Occurs when Milkyway successfully retrieves the library attached files.
MWXFail	Occurs when Milkyway fails to retrieve the library attached files.

Description

The number of attached files is stored in "attachedFileCnt." The point of pointer of file names is stored in "attachedFiles." The point of pointer of pseudo names is stored in "pseudoAccessNames."

For a cell of CEL view, the attached file name string appears similar to the following:

```
"CEL/killUnit:2_1"
```

For a cell of LOGIG/LM view, the attached file name string appears similar to the following:

```
"LOGIC/design_1.db"
```

Note:

It is your responsibility to free the allocated memory of string pointer.

Example

```
void Print_LibraryAttachedFiles(
    MWXLibId_t libId,
{
    int32 attachedFileCnt;
    int k;
    MWXDBString_t *attachedFiles, *pseudoAccessNames;
    int32 *fileExistStatus;
    if (MWXDb_Get_LibAttachedFiles(libId,&attachedFileCnt,
        &attachedFiles,
        &pseudoAccessNames,&fileExistStatus) !=
        MWXSuccess) {
        MWX_Printf("FAIL to get library attached files\n");
        exit(MWXFail);
    }
    for(k = 0; k < attachedFileCnt; k++) {
        MWX_Printf("SUCCESS get the attached files. Their
            info are:\n");
        MWX_Printf("pseudoAccessNames \t attachedFiles \t
            fileExistStatus\n");
        MWX_Printf("%s \t %s \t
            %d\n",pseudoAccessNames[k],attachedFiles[k],
            fileExistStatus[k]);
    }
    if (pseudoAccessNames) {
        free(pseudoAccessNames);
    }
    if (attachedFiles) {
        free(attachedFiles);
    }
    if (fileExistStatus) {
        free (fileExistStatus);
    }
}
```

MWXDb_Get_LibId_ByCellId

Gets the database identifier of the library that contains the specified cell.

Syntax

```
int MWXDb_Get_LibId_ByCellId (
    MWXCellId_t cellId, /*INPUT*/
    MWXLibId_t *libId /*OUTPUT*/
)
```

Parameters

cellId	Database identifier of any open cell.
libId	Database identifier of the library that contains the specified cell. This can be the parent or one of the reference libraries.

Return

MWXFail	Occurs if the library identifier is not successfully retrieved.
MWXSuccess	Occurs if the library identifier is successfully retrieved.

Example

```
rc = MWXDb_Get_LibId_ByCellId(cellId, &libId);
switch (rc) {
    case MWXSuccess:
        return (MWXSc_Make_DBIIdSCM (libId));
    case MWXFail:
        return (MWXFail);
    default:
        return (MWXFail);
}
```

Scheme Equivalent

dbGetLibIdByCellId
(see Physical Implementation Online Help)

MWXDb_Get_LibId_ByLibName

Finds the library ID for a specified full library name.

Syntax

```
int MWXDb_Get_LibId_ByLibName (
    char libName, /* INPUT */
    MWXLibId_t *libId /* OUTPUT */
)
```

Parameters

libName	Specifies the full library name (absolute path or relative path).
libId	Returned database identification number of the library.

Return

MWXSuccess	Occurs when the command successfully gets the library ID by using the library name.
MWXFail	Occurs when the command fails to get the library ID.

Dynamic Example

```
#include<stdio.h>
#include<stdlib.h>
#include "mwX.h"
#include "mwX_proto.h"

MWXSCM_t mwGetLibIdByLibName (MWXSCM_t list)
{
    char progDescrString[50] = "Get LibId By LibName -";
    char libName[256];
    MWXLibId_t libId;
    static int caseSensitive = 0;
    /*-----
       GET THE SCHEME LIST
       -----*/
    if (MWXSc_Scanf (list, "s",libName) != 1) {
        MWX_Printf((char*)"Error.Synopsis : getLibIdByLibName libName \n");
        return (MWXFail);
    }
    /*-----
       HELLO
       -----*/
    MWX_Printf((char*)" \n%s Testing...\n", progDescrString);

    /* Get libId by library name */

    if (MWXDb_Get_LibId_ByLibName(libName, &libId) != MWXSuccess) {
        MWX_Printf((char*) "Error : Can't get LibId by libName\n");
        return(MWXFail);
    }

    MWX_Printf((char *) "LibId of lib %s is %d\n",libName, libId);
    /*-----
       DONE
       -----*/
    MWX_Printf((char*)" %s DONE TESTING! \n\n",progDescrString);
    return(MWXSuccess);
}

void reg_getLibIdByLibName(){
```

```

        MWXSc_Register_Function((char*)"getLibIdByLibName",mwGetLibIdByLibName);
    }

```

Stand-Alone Example

```

int GetLibIdByLibName (char *libName)
{
    MWXLibId_t libId;

    /* Get libId by library name */

    if (MWXDb_Get_LibId_ByLibName(libName, &libId) != MWXSuccess) {
        MWX_Printf((char*) "Error : Can't get LibId by libName\n");
        return(MWXFail);
    }

    MWX_Printf((char *) "LibId of lib %s is %d\n",libName, libId);
    return(MWXSuccess);
}

```

See Also

[MWXDb_Get_LibId_ByCellId](#)

MWXDb_Get_LibHierDelimiter

Retrieves the hierarchy delimiter used by the library with the given library ID.

Syntax

```

int MWXDb_Get_LibHierDelimiter(
    MWXLibId_t libId /* INPUT */
    uchar *delimiter /* OUTPUT */
)

```

Parameters

libId	Specifies the library ID of the library you want to query.
delimiter	Returns the hierarchy delimiter used by the given library.

Return

MWXSuccess	Occurs if the delimiter is successfully retrieved.
MWXFail	Occurs if the delimiter is not retrieved.

Dynamic Example

```

MWXSCM_t mwGetLibHierDelimiter(MWXSCM_t list) {

```

```

MWXLibId_t libId;
uchar delimiter;
int rc;

if (MWXSc_Scanf(list, "o", &libId) != 1) {
    MWX_Printf("ERROR - Usage: mwGetLibHierDelimiter libId\n");
    return MWX_FALSE_SCM;
}

rc = MWXDb_Get_LibHierDelimiter(libId, &delimiter);

switch ( rc ) {
case MWXSuccess:
    MWX_Printf("Library's hierarchy delimiter: '%c'\n", delimiter);
    return MWX_TRUE_SCM;
case MWXFail:
    MWX_Printf("Failed to retrieve library hierarchy delimiter.\n");
    return MWX_FALSE_SCM;
default:
    MWX_Printf("ERROR -- Unknown return code %d. MWXDb_Get_LibHierDelimiter "
               "failed\n", rc);
    return MWX_FALSE_SCM;
}
}

```

Stand-Alone Example

```

int GetLibHierDelimiter(MWXLibId_t libId, uchar *delimiter) {

    int rc;

    MWX_Printf("INFO -- MWXDb_Get_LibHierDelimiter.\n");
    rc = MWXDb_Get_LibHierDelimiter (libId, delimiter);

    switch (rc) {
case MWXSuccess:
    MWX_Printf("Library's hierarchy delimiter: '%c'\n", *delimiter);
    return MWXSuccess;
case MWXFail:
    MWX_Printf("Failed to retrieve library hierarchy delimiter.\n");
    return MWXFail;
default:
    MWX_Printf("ERROR -- Unknown return code %d. MWXDb_Get_LibHierDelimiter "
               "failed\n", rc);
    return MWXFail;
}
}

```

MWXDb_Get_LibName_ById

Get the library name for the ID of the opened library.

Syntax

```
int MWXDb_Get_LibName_ById(  
    MWXLibId_t libId,      /* INPUT */  
    int32 pathMode,        /* INPUT */  
    char **libName         /* OUTPPUT */  
)
```

Parameters

libId	Database identifier for the Milkyway library
pathMode	Indicates whether you want to output the full path name of the library or just the name itself. Valid values: 1 - Output the full path name of library 2 - Output just the library name
libName	Returned Library Name

Return

MWXSuccess	Occurs if successfully gets the library name.
MWXFail	Occurs if fails to get the library name

Example

```
    MWXLibId_t libId = 1;  
char libName = NULL;  
int32 path_mode = 0;  
  
rc = MWXDb_Get_LibName_ById (libId, path_mode, &libName);  
  
switch (rc) {  
    case MWXSuccess:  
        MWX_Printf("Successfully get libName %s\n", libName);  
        return 1;  
  
    case MWXFail:  
        MWX_Printf("Fail to get library name of libId = %d\n",libId);  
        return 0;  
    default:  
        MWX_Printf("Unknown return value\n");  
        return 0;  
}
```

Scheme Equivalent

dbGetLibNameById

MWXDb_Get_LibViewNames

Returns a list of names for every view that has a cell in the library.

Syntax

```
public int
MWXDb_Get_LibViewNames(
    MWXLibId_t libId /* INPUT */
    int32 **viewCount /* OUTPUT */
    MWXDBString_t **viewNames /* OUTPUT */
)
```

Parameters

libId	Specifies the design library database identification.
viewCount	Specifies the total retrieved view names.
viewNames	Specifies the list of retrieved view names.

Return

MWXNoLibraryOpened	Occurs if no library opens.
MWXSuccess	Occurs if the list of view names is successfully returned.
MWXFail	Occurs if Milkyway fails to return the list of view names.

Example

```
void Print_Cells(
    MWXLibId_t libId )
{
    MWXDBString_t *viewNames, *cellNames;
    int32 viewCount, cellCount, latestVersion;
    int rc, i , j;
    viewCount = 0;
    MWX_Printf("Trying to get all viewNames in library \n");
    rc = MWXDb_Get_LibraryViewNames(libId,
        &viewCount,&viewNames;
    if (rc != MWXSuccess)
        MWX_Printf("Fail to get library viewNames\n");
        exit (MWXFail);
    } else {
    for (i=0; i<viewCount; i++) {
        cellCount = 0;
        rc = MWXDb_Get_CellNames_ByView(libId,
            viewNames[i], &cellCount, &cellNames);
        if (rc == MWXSuccess) {
```

```

        for (j = 0; j < cellCount; j++) {
            latestVersion = 0;
            if (MWXDb_Get_CellLatestVersion(libId,
                cellNames[j],viewNames[i],
                &latestVersion) != MWXSuccess) {
                MWX_Printf("FAIL - Unable to get
                           the cell
                           latest version\n");
                free_dbString_memory(cellNames);
                free_dbString_memory(viewNames);
                exit (MWXFail);
            }
            MWX_Printf("%s Finish testing get
cellLatestVersion\n",progDescrString);
            if (MWXDb_Get_CellFullName(libId,
                cellNames[j],viewNames[i],
                latestVersion,&cellFullName) !=
                MWXSuccess) {
                MWX_Printf("FAIL - Unable to get the cell
                           full name\n");
                free_dbString_memory(cellNames);
                free_dbString_memory(viewNames);
                exit (MWXFail);
            }
            MWX_Printf("Cell Full Name is
                       %s\n",cellFullName);
        }
    }
    MWX_Printf("%s Finish testing the cell full
name\n",progDescrString);}
}

```

MWXDb_Get_LogicModelSubType

Retrieves the logic model subtype for a given cell, including any latch, flip-flop, RAM, ROM, or clock cell in the library.

Syntax

```

int MWXDb_Get_LogicModelSubType(
    MWXCellId_t cellId, /* INPUT */
    int *subType        /* OUTPUT */
)

```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell that was opened.
<code>subType</code>	Specifies the subtype of a cell that is going to be returned. Returned values: 1 - flip-flop 2 - clock 3 - latch 4 - RAM 5 - ROM 0 - not assigned

Return

<code>MWXFail</code>	Occurs when the cell subtype is not successfully retrieved.
<code>MWXSuccess</code>	Occurs when the cell subtype is successfully retrieved.

Example

```
rc = MWXDb_Get_LogicModelSubType (cellId, &subType);
switch (rc) {
    case MWXSuccess:
        break;
    case MWXFail:
        MWX_Printf("\nERROR -- Failed to retrieve cell
            subtype!\n");
        return (MWXFail);
    default:
        MWX_Printf("\nERROR -- Unknown return code!\n");
        return (MWXFail);
}
```

Scheme Equivalent

`dbGetLModelSubType`
(see Physical Implementation Online Help)

MWXDb_Get_MaskNames

Gets the layer numbers and corresponding mask names with the specified library identity. You must free the memory that is allocated for `layerNumbers` and `maskNames` when the data is no longer needed.

Syntax

```
int MWXDb_Get_MaskNames (
    MWXLibId_t libId, /* INPUT */
    int *count, /* OUTPUT */
    int **layerNumbers, /* OUTPUT */
    MWX_tfName_t **maskNames /* OUTPUT */
)
```

Parameters

libId	Specifies the database identifier of the library from which to get mask names.
count	Specifies the number of mask names found in this library.
layerNumbers	Specifies an array of layer numbers of each mask name.
maskNames	Specifies an array of mask names.

Return

MWXSuccess	Occurs if the mask names are successfully retrieved.
MWXFail	Occurs if the mask names are not successfully retrieved.

Example

```
rc = MWXDb_Get_MaskNames (libId, &count, &layerNumbers, &maskNames);

switch ( rc ) {
    case MWXSuccess:
        MWX_Printf("there are %d mask names in this library.\n", count);
        for ( i = 0 ; i < count ; i ++ )
            MWX_Printf("    layer number : %d, mask name : %s \n",
                *(layerNumbers + i), maskNames + i);

        /* Don't forget to free the memories. */
        if ( layerNumber != NULL )
            free( layerNumber );
        if ( maskNames != NULL )
            free ( maskNames );
        break;
    case MWXFail:
        MWX_Printf("failed to get mask names.\n");
        return MWXFail;
}

return MWXSuccess;
```

MWXDb_Get_Net_ByName

Retrieves the database identifier for the net with the name `netName`. The database identifier of the net is returned in `netId`.

Note:

`netId` is undefined if the return code is not `MWXSuccess`.

Syntax

```
int MWXDb_Get_Net_ByName (
    MWXCellId_t cellId, /* INPUT */
    char *netName, /* INPUT */
    MWXObjId_t *netId /* OUTPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell containing the net.
<code>netName</code>	Specifies the name of the net to retrieve.
<code>netId</code>	Specifies the database identifier of the net with the name <code>netName</code> .

Return

<code>MWXSuccess</code>	Occurs if the net is successfully found.
<code>MWXFail</code>	Occurs if the net is not successfully found.

Example

```
rc = MWXDb_Get_Net_ByName (cellId, netName, &netId);
switch (rc) {
    case MWXSuccess:
        break;
    case MWXFail:
        MWX_Printf("ERROR -- Failed to get net\n");
        break;
}
```

MWXDb_Get_NetComponents

Retrieves database identifiers for all the objects attached to a net. The `MWXDb_Get_NetComponents` function allocates memory for `objectsIdArray`. It is your responsibility to recover this memory when you no longer need the data.

Syntax

```
int MWXDb_Get_NetComponents(  
    MWXCellId_t cellId, /*INPUT*/  
    MWXObjId_t netId, /*INPUT */  
    int *objectCount, /*OUTPUT */  
    MWXObjId_t**objectsIdArray /*OUTPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell from which to get the net components.
netId	Specifies the database identifier of the net from which to get the components.
objectCount	Specifies the number of components found on the net and returned in objectsIdArray.
objectsIdArray	Specifies an array of database identifiers for the net components on the net.

Return

MWXSuccess	Occurs if the objects are successfully retrieved.
MWXFail	Occurs if the objects are not successfully counted (note that if no objects are attached to the net, zero objects are returned and it is considered a successful retrieval).

Example

```
rc = MWXDb_Get_NetComponents (cellId, netId, &objectCount, &objectsIdArray);  
    switch (rc) {  
        case MWXSuccess:  
            break;  
        case MWXFail:  
            MWX_Printf("ERROR -- failed to get net  
                        objects\n");  
            break;  
    }
```

Scheme Equivalent

db-foreach

(see Physical Implementation Online Help)

MWXDb_Get_NetPort

Retrieves port database identifiers for all the ports to which the net is attached. These are the ports of the parent cell of the net, not port instances of the cell instantiations. The `MWXDb_Get_NetPort` function allocates memory for `objectsIdArray`. It is your responsibility to recover this memory when you no longer need the data.

Syntax

```
int MWXDb_Get_NetPort(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t netId, /* INPUT */  
    int *objectCount, /* OUTPUT */  
    MWXObjId_t **objectsIdArray /* OUTPUT */  
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell from which to get the net ports.
<code>netId</code>	Specifies the database identifier of the net from which to get the ports.
<code>objectCount</code>	Specifies the number of ports to which the net is attached and returned in <code>objectsIdArray</code> .
<code>objectsIdArray</code>	Specifies an array of database identifiers for the ports to which the net is attached.

Return

<code>MWXSuccess</code>	Occurs if the ports are successfully retrieved.
<code>MWXFail</code>	Occurs if the ports are not successfully retrieved (note that if no ports are attached to the net, zero objects are returned and it is considered a successful retrieval).

Example

```
rc = MWXDb_Get_NetPort (cellId, objectType, &objectCount, &objectsIdArray);  
    switch (rc) {  
        case MWXSuccess:  
            break;  
        case MWXFail:  
            MWX_Printf("ERROR -- failed to get ports to  
                which net is attached\n");  
            break;  
    }
```


Scheme Equivalent

db-foreach

(see Physical Implementation Online Help)

MWXDb_Get_NetPortInst

Retrieves port instance database identifiers for all the port instances attached to a net. Note that `MWXDb_Get_NetPortInst` allocates memory for `objectsIdArray`. You are responsible for recovering this memory when you no longer need the data.

Syntax

```
int MWXDb_Get_NetPortInst(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t netId, /* INPUT */  
    int *objectCount, /* OUTPUT */  
    WXObjId_t **objectsIdArray /* OUTPUT */  
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell from which to get port instances attached to a net.
<code>netId</code>	Specifies the database identifier of the net for which attached port instances are retrieved.
<code>objectCount</code>	Specifies the number of port instances attached to the net and returned in <code>objectsIdArray</code> .
<code>objectsIdArray</code>	Specifies an array of database identifiers for the port instances attached to the net.

Return

<code>MWXSuccess</code>	Occurs if the port instances are successfully retrieved.
<code>MWXFail</code>	Occurs if the port instances are not successfully retrieved (note that if no ports are attached to the net, zero objects are returned and it is considered a successful retrieval)

Example

```
rc = MWXDb_Get_NetPortInst (cellId, objectType,  
    &objectCount,&objectsIdArray);  
switch (rc) {  
    case MWXSuccess:  
        break;
```

```

        case MWXFail:
            MWX_Printf("ERROR -- failed to get port
            instances attached to net\n");
            break;
    }

```

Scheme Equivalent

db-foreach
(see Physical Implementation Online Help)

MWXDb_Get_NetWiring

Retrieves database identifiers for all of the wire objects attached to a net. The objects can be of the following types:

- MWX_HORIZONTALWIRE
- MWX_VERTICALWIRE
- MWX_CONTACTARRAY
- MWX_PATH

Use the `MWXDb_Get_ObjectsType` function to determine the type of each object. Note that `MWXDb_Get_NetComponents` allocates memory for `objectsIdArray`. It is your responsibility to recover this memory when you no longer need the data.

Syntax

```

int MWXDb_Get_NetWiring(
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t netId,   /* INPUT */
    int *objectCount,   /* OUTPUT */
    MWXObjId_t**objectsIdArray /* OUTPUT */
)

```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell from which to get net wire objects.
<code>netId</code>	Specifies the database identifier of the net from which to get wire objects.
<code>objectCount</code>	Specifies the number of wire objects found on the net and returned in <code>objectsIdArray</code> .
<code>objectsIdArray</code>	Specifies an array of database identifiers for the wire objects on the net.

Return

<code>MWXSucces</code>	Occurs if the objects are successfully retrieved.
<code>MWXFail</code>	Occurs if the objects are not successfully counted (note that if no objects are attached to the net, zero objects are returned and it is considered a successful retrieval).

Example

```
rc = MWXDb_Get_NetWiring (cellId, netId, &objectCount, &objectsIdArray);
    switch (rc) {
        case MWXSucces:
            break;
        case MWXFail:
            MWX_Printf("ERROR -- failed to get net
            objects\n");
            break;
    }
```

Scheme Equivalent

`db-foreach`

(see Physical Implementation Online Help)

MWXDb_Get_Objects_ByBBox

Retrieves all objects of all layers of the specified objectType where the bbox of the object overlaps with the specified bbox. It is your responsibility to recover the memory allocated for the `objectsIdArray` field when the data is no longer needed.

Syntax

```
int32 MWXDb_Get_Objects_ByBBox(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjTypeId_t objectType, /* INPUT */  
    int32 left, /* INPUT */  
    int32 bottom, /* INPUT */  
    int32 right, /* INPUT */  
    int32 top, /* INPUT */  
    int32 *objectCount, /* OUTPUT */  
    MWXObjId_t **objectsIdArray /* OUTPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell that was opened.
objectType	The only type of the object you want to get.
left	Specifies the x-coordinate for the left side of the bounding box.
bottom	Specifies the y-coordinate for the bottom of the bounding box.
right	Specifies the x-coordinate for the right side of the bounding box.
top	Specifies the y-coordinate for the top of the bounding box.
objectCount	The number of the objects in the input.
objectsIdArray	An array of all object IDs. It is your responsibility to free the memory—for example, <code>free(*objectsIdArray)</code> .

Return

MWXSuccess	Occurs if objects are successfully retrieved from the bbox.
MWXFail	Occurs if the objects are not successfully retrieved from the bbox.

Example

```
if (MWXDb_Get_ObjectsType(cellId, objId, &objType) != MWXSuccess) {  
    MWX_Printf("Invalid object Id. Please try again! \n");  
    return (MWXFail);  
}  
if (MWXDb_Get_Objects_ByBBox (cellId, objType, left, bottom, right, top,  
    &objCount, &objsIdArray) != MWXSuccess) {  
    MWX_Printf("Can't get objects by bbox, invalid data\n");  
    return (MWXFail);  
}  
if (objCount > 0) free (objsIdArray);
```

```
return (MWXSc_Make_IntegerSCM(objCount));
```

MWXDb_Get_Objects_ByBBoxSearch

Retrieves all objects of a specified layer where the shape of the objects overlaps the specified bbox. It is your responsibility to recover the memory allocated for the `objectsIdArray` field when the data is no longer needed.

Note:

When `layer = 0`, it applies to all layers. Cell instances are not included in the objects search. If you attempt to make a sequence of this function, call [MWXDb_Startup_ShapeStruct](#) first.

Syntax

```
int MWXDb_Get_Objects_ByBBoxSearch(  
    MWXCellId_t cellId, /*INPUT*/  
    int32 layer, /* INPUT */  
    int32 left, /* INPUT */  
    int32 bottom, /* INPUT */  
    int32 right, /* INPUT */  
    int32 top, /* INPUT */  
    int32 *objectCount, /* OUTPUT */  
    MWXObjId_t **objectsIdArray /* OUTPUT */  
)  
{
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell that was opened.
<code>layer</code>	Specifies the layer; <code>layer == 0</code> means all layers.
<code>left</code>	Specifies the x-coordinate of the left side of the bounding box.
<code>bottom</code>	Specifies the y-coordinate of the bottom of the bounding box.
<code>right</code>	Specifies the x-coordinate of the right side of the bounding box.
<code>top</code>	Specifies the y-coordinate of the top of the bounding box.
<code>objectCount</code>	Specifies the number of the objects being returned in <code>objectsIdArray</code> .
<code>objectsIdArray</code>	Specifies the array of database identifiers for objects retrieved.

Return

<code>MWXSuccess</code>	Occurs if objects are successfully retrieved by the bbox search.
<code>MWXFail</code>	Occurs if the objects are not successfully retrieved by the bbox search.

Example

```
if (MWXDb_Get_Objects_ByBBoxSearch (cellId, layer, left, bottom, right, top,
&objCount, &objsIdArray) != MWXSuccess) {
    MWX_Printf("Can't get objects by bbox, invalid data\n");
    return (MWXFail);
}
if (objCount > 0) free (objsIdArray);
return (MWXSc_Make_IntegerSCM(objCount));
```

See also:

[MWXDb_Cleanup_ShapeStruct](#)
[MWXDb_Startup_ShapeStruct](#)

MWXDb_Get_Objects_ByType

Retrieves the database identifiers for all objects of type `objectType` that are contained in the cell identified by `cellId`. The function allocates memory for the object array. It is your responsibility to recover the memory when you no longer need the data.

Syntax

```
int MWXDb_Get_Objects_ByType(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjTypeId_t objectType, /* INPUT */  
    int *objectCount, /* OUTPUT */  
    MWXObjId_t **objectsIdArray /* OUTPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell containing objects to retrieve.
objectType	Specifies the type name (from mwx.h) of the objects to retrieve.
objectCount	Specifies the number of objects being returned in objectsIdArray.
objectsIdArray	Specifies the array of database identifiers for the objects retrieved.

Return

MWXSuccess	Occurs if the objects are successfully retrieved.
MWXFail	Occurs if the objects are not successfully retrieved (if no objects of the requested type exist, the retrieval is still considered successful and zero objects are returned).

Example

```
rc = MWXDb_Get_Objects_ByType (cellId, objectType,  
    &objectCount,&objectsIdArray);  
    switch (rc) {  
        case MWXSuccess:  
            break;  
        case MWXFail:  
            MWX_Printf("ERROR -- Failed to get  
                objects\n");  
            break;  
    }  
}
```

Scheme Equivalent

db-foreach

(see Physical Implementation Online Help)

MWXDb_Get_Objects_ByTypeAndLayer

Retrieves the database identifiers for all objects of type `objType` contained in the cell identified by `cellId` on a specific layer. The function allocates memory for the object array. It is your responsibility to recover the memory when you no longer need the data.

Syntax

```
int MWXDb_Get_Objects_ByTypeAndLayer(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjTypeId_t objType, /* INPUT */  
    int32 layer, /* INPUT */  
    int32 *objectCount, /* OUTPUT */  
    MWXObjId_t **objectsIdArray /* OUTPUT */  
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell containing the objects to retrieve.
<code>objType</code>	Specifies the type name (from <code>mwX.h</code>) of the objects to retrieve.
<code>layer</code>	Specifies the specific layer from which the objects are being retrieved.
<code>objectCount</code>	Specifies the number of objects being returned in <code>objectsIdArray</code> .
<code>objectsIdArray</code>	Specifies the array of database identifiers for the objects retrieved.

Return

<code>MWXSuccess</code>	Occurs if the objects are successfully retrieved.
<code>MWXFail</code>	Occurs if the objects are not successfully retrieved (if no objects of the requested type on a given layer exist, the retrieval is still considered successful and zero objects are returned).

Example

```
rc = MWXDb_Get_Objects_ByTypeAndLayer (cellId, objectType, layerNumber,  
    &objectCount,&objectsIdArray);  
    switch (rc) {  
        case MWXSuccess:  
            break;  
        case MWXFail:  
            MWX_Printf("ERROR -- Failed to get
```



```

        objects\n");
        break;
    }

```

MWXDb_Get_ObjectsCount

Counts the number of objects of type `objectType` found in the cell identified by `cellId`. The count is returned in `objectCount`. The object types are listed in the `mw.x.h` file.

Syntax

```

int MWXDb_Get_ObjectsCount(
    MWXCellId_t cellId, /* INPUT */
    MWXObjTypeId_t objectType, /* INPUT */
    int *objectCount /* OUTPUT */
)

```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell in which to count objects.
<code>objectType</code>	Specifies the type of the objects to count.
<code>objectCount</code>	Specifies the number of the objects of type <code>objectType</code> found in the cell.

Return

<code>MWXSuccess</code>	Occurs if the objects are successfully counted.
<code>MWXFail</code>	Occurs if the objects are not successfully counted (note that a count of zero is still a successful count).

Example

```

rc = MWXDb_Get_ObjectsCount (cellId, objectType, &objectCount);
    switch (rc) {
        case MWXSuccess:
            break;
        case MWXFail:
            MWX_Printf("ERROR -- failed to count
                objects\n");
            break;
    }

```

MWXdB_Get_ObjectsType

Determines the type of an object. The cell containing the object must be open for successful operation.

Syntax

```
int MWXdB_Get_ObjectsType (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t objectId, /* INPUT */
    MWXObjTypeId_t *objectType /* OUTPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that owns the object defined by <code>objectId</code> .
objectId	Specifies the database identifier of object for which the type is being retrieved.
objectType	Specifies the type of the object defined by <code>objectId</code> (object types are defined in the <code>mwX.h</code> file).

Return

MWXSuccess	Occurs if the object type is successfully retrieved.
MWXFail	Occurs if the object type retrieval fails.

Example

```
rc = MWXdB_Get_ObjectsType (cellId, objectId, &objectType);
    if (MWXSuccess) {
        return (MWXSc_Make_StringSCM
            (ObjectName(objectType)));
    } else {
        return (MWXFail);
    }
```

MWXdB_Get_OwnerLibId_ByCellName

Searches the library tree and gets the identification number of the library that owns the specified cell. The search starts with the given library ID.

Syntax

```
int MWXDb_Get_OwnerLibId_ByCellName (
    MWXLibId_t mainLibId, /* INPUT */
    char *cellName, /* INPUT */
    MWXLibId_t *ownerLibId /* OUTPUT */
)
```

Parameters

mainLibId	Specifies the library ID of the first library searched in the hierarchical tree.
cellName	Specifies the cell name that is used to search for the owner library ID.
ownerLibId	Specifies the returned library ID of the owner library of the cell.

Return

MWXSuccess	Occurs if the owner library ID is successfully retrieved.
MWXNoLibraryOpened	Occurs if no library is opened.
MWXFail	Occurs if the owner library ID is not successfully retrieved.

Dynamic Example

```
#include<stdio.h>
#include<stdlib.h>
#include "mwX.h"
#include "mwX_proto.h"

MWXSCM_t mwGetOwnerLibIdByCellName (MWXSCM_t list)
{
    char progDescrString[50] = "Get OwnerLibId by cellName -";
    char cellName[256];
    char *libName;
    MWXLibId_t mainLibId, libId;
    static int caseSensitive = 0;
    /*-----
    GET THE SCHEME LIST
    -----*/
    if (MWXSc_Scanf (list, "os",&mainLibId,cellName) != 2) {
        MWX_Printf((char*)"Error.Synopsis : getOwnerLibIdBycellName mainLibId
cellName\n");
        return (MWXFail);
    }
    /*-----
    HELLO
    -----*/
    MWX_Printf((char*)" \n%s Testing...\n", progDescrString);
```

```

    if (MWXDb_Get_OwnerLibId_ByCellName(mainLibId,cellName, &libId) != MWXSuccess)
    {
        MWX_Printf((char*) "Error : Can't get ownerLibId by cellName %s\n",cellName);
        return (MWXFail);
    }

    if (MWXDb_Get_LibName_ById(libId,0, &libName) != MWXSuccess) {
        MWX_Printf((char*) "Error : Can't get libName by libId = %d\n",libId);
        return (MWXFail);
    }

    /*-----
    DONE
    -----*/
    MWX_Printf((char*)"OwnerLibName of cell %s is %s\n",cellName, libName);
    MWX_Printf((char*)" %s DONE TESTING! \n\n",progDescrString);
    return (MWXSuccess);
}
void reg_getOwnerLibIdByCellName(){
MWXSc_Register_Function((char*)"getOwnerLibIdByCellName",mwGetOwnerLibIdByCell
Name);
}

```

Stand-Alone Example

```

#include<stdio.h>
#include<stdlib.h>
#include "mwx.h"
#include "mwx_proto.h"
#include "mda_proto.h"

int GetOwnerLibIdByCellName(
    MWXLibId_t mainLibId,
    char *cellName
)
{
    char *libName;
    MWXLibId_t libId;

    if (MWXDb_Get_OwnerLibId_ByCellName(mainLibId,cellName, &libId) != MWXSuccess)
    {
        MWX_Printf((char*) "Error : Can't get ownerLibId by cellName
%s\n",cellName);
        return(MWXFail);
    }

    if (MWXDb_Get_LibName_ById(libId,0, &libName) != MWXSuccess) {
        MWX_Printf((char*) "Error : Can't get libName by libId = %d\n",libId);
        return (MWXFail);
    }

    MWX_Printf((char*)"OwnerLibName of cell %s is %s\n",cellName, libName);
    return (MWXSuccess);
}

```

See Also

MWXDb_Check_Cell_Exist
MWXDb_Check_Cell_ExistInLibrary

MWXDb_Get_Pin_ByName

Retrieves the database identifier for the pin with the name `pinName`. The database identifier of the pin is returned in `pinId`. The `pinId` is undefined if the return code is not `MWXSuccess`. Also, if there is more than one pin with the name `pinName`, `pinId` will point to an array of database identifiers.

Syntax

```
int MWXDb_Get_Pin_ByName(  
    MWXCellId_t cellId, /* INPUT */  
    char *pinName, /* INPUT */  
    MWXObjId_t *pinId /* OUTPUT */  
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell containing the pin.
<code>pinName</code>	Specifies the name of the pin to retrieve.
<code>pinId</code>	Specifies the database identifier of the pin with name <code>pinName</code> .

Return

<code>MWXSuccess</code>	Occurs if the pin is successfully found.
<code>MWXFail</code>	Occurs if the pin is not successfully found.

Example

```
rc = MWXDb_Get_Pin_ByName (cellId, pinName, &pinId);  
switch (rc) {  
    case MWXSuccess:  
        break;  
    case MWXFail:  
        MWX_Printf("ERROR -- Failed to get pin\n");  
        break;  
}
```

Scheme Equivalent

`dbGetPinByName` (see *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Get_PinFromPort

Retrieves pin database identifiers for all pins to which the port is attached. Note that `MWXDb_Get_PinFromPort` allocates memory for `objectsIdArray`. It is your responsibility to recover this memory when this data is no longer needed.

Syntax

```
int MWXDb_Get_PinFromPort (
    MWXCellId_t cellId, /*INPUT*/
    MWXObjId_t portId, /*INPUT*/
    int *objectCount, /*OUTPUT*/
    MWXObjId_t **objectsIdArray /*OUTPUT*/
)
```

Parameters

<code>cellId</code>	Database identifier of the cell that contains the port.
<code>portId</code>	Database identifier of the port.
<code>objectCount</code>	The number of pins to which the port is attached.
<code>objectsIdArray</code>	An array of database identifiers for the pins to which the port is attached.

Return

<code>MWXSuccess</code>	Occurs if the pins are successfully retrieved.
<code>MWXFail</code>	Occurs if the pins are not successfully retrieved. Note that if no pins are attached to the port, zero objects are returned, and it is considered a successful retrieval.

Example

```
rc = MWXDb_Get_PinFromPort (cellId, portId,
    &objectCount, &objectsIdArray);

switch (rc) {
    case MWXSuccess:
        if (objectCount > 0) free (objectsIdArray);
        return (MWXSc_Make_IntegerSCM (objectCount));
    case MWXFail:
        return (MWXFail);
    default:
        return (MWXFail);
}
```

MWXdB_Get_PinInstMaster_numberOfPoint

Retrieves the data for the numberOfPoint field in the PinInstMaster record.

Syntax

```
int32 MWXdB_Get_PinInstMaster_numberOfPoint(  
    MWXCellId_t cellId, /*INPUT*/  
    MWXObjId_t objId, /*INPUT*/  
    int32 *value, /*OUTPUT*/  
)
```

Parameters

cellId	Specifies the database identifier of the cell that contains the objects to retrieve.
objId	Specifies the database identifier of the object to retrieve. The object's type must be AX_PinInstMaster_tId.
value	Specifies the value of the numberOfPoint field.

Return

MWXSuccess	Occurs if you get the numberOfPoint PinInstMaster record.
MWXFail	Occurs if you fail to get the numberOfPoint PinInstMaster record.

Example

```
rc = MWXdB_Get_PinInstMaster_name (cellId, objId,  
    &value);  
switch (rc) {  
    case MWXSuccess:  
        break;  
    case MWXFail:  
        MWX_Printf("ERROR -- Failed to get  
        numberOfPoint\n");  
        break;  
}
```

MWXdB_Get_PinNet_ByName

This function retrieves the pin's netId for a given pinName.

Syntax

```
int MWXDb_Get_PinNet_ByName(  
MWXCellId_t cellId, /* INPUT */  
char *pinName, /* INPUT */  
MWXObjId_t *netId /* OUTPUT */  
)
```

Parameters

cellId	Specifies the cellId in the database
pinName	Name of the pin that you want to get the net info from

Return

MWXSuccess	Occurs if successfully get the pinNet by pin Name
MWXFail	Occurs if fail to get the pin Net by pin name

Example

```
rc = MWXDb_Get_PinNet_ByName(cellId, pinName,&netId);  
switch (rc) {  
    case MWXSuccess:  
        return (netId);  
    case MWXFail:  
        printf("Fail to retrieve pinNet by name\n");  
        return 0;  
    Default:  
        printf("Unknown return code \n");  
        return 0;  
}
```

MWXDb_Get_PlanGroup_ByName

Retrieves the floorplan group database identifier.

Syntax

```
int MWXDb_Get_PlanGroup_ByName (  
    MWXCellId_t cellId, /* INPUT */  
    char *groupName, /* INPUT */  
    MWXObjId_t *group /* OUTPUT */  
)
```


Parameters

<code>cellId</code>	Specifies the database identifier of the cell that owns the floorplan group.
<code>groupName</code>	Specifies the name of the group.
<code>groupId</code>	Database identifier of the floorplan group.

Return

<code>MWXNoLicense</code>	Occurs if no software license is available.
<code>MWXSuccess</code>	Occurs if the database identifier of the group is successfully retrieved.
<code>MWXFail</code>	Occurs if the database identifier of the group is not successfully retrieved.

Example

```
rc = int MWXDb_Get_PlanGroup_ByName( cellId, groupName,&groupId )
    switch (rc) {
        case MWXSuccess:
            return (MWXSuccess);
        case MWXFail:
            return (MWXFail);
    }
```

Scheme Equivalent

`dbGetPlanGroupByName`
(see Physical Implementation Online Help)

MWXDb_Get_Port_ByName

Retrieves the database identifier for the port with the name `portName`. The database identifier of the port is returned in `portId`, which is undefined if the return code is not `MWXSuccess`.

Syntax

```
int MWXDb_Get_Port_ByName(
    MWXCellId_t cellId, /* INPUT */
    char *portName, /* INPUT */
    MWXObjId_t *portId /* OUTPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell containing the port.
portName	Specifies the name of the port to retrieve.
portId	Specifies the database identifier of the port with the name portName.

Return

MWXSuccess	Occurs if the port is successfully found.
MWXFail	Occurs if the port is not successfully found.

Example

```
rc = MWXDb_Get_Port_ByName (cellId, portName, &portId);
    switch (rc) {
        case MWXSuccess:
            break;
        case MWXFail:
            MWX_Printf("ERROR -- Failed to get port\n");
            break;
    }
```

Scheme Equivalent

dbGetPortByName

(see Physical Implementation Online Help)

MWXDb_Get_PortInstFromCellInst

Retrieves the database identifier for the port instances attached to a cell instance. The output parameters of the function are undefined if the function does not return `MWXSuccess`. It is your responsibility to free the allocated memory in the `objectsIdArray` field when the data is no longer needed.

Syntax

```
FILE *MWXDb_Get_PortInstfromCellInst(
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t cellInstId, /* INPUT */
    int *objectCount, /* OUTPUT */
    MWXObjId_t **objectsIdArray /* OUTPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell containing the cell instance.
cellInstId	Specifies the database identifier of the cell instance whose port instances are being retrieved.
objectCount	Specifies the number of port instances returned in objectsIdArray.
objectsIdArray	Specifies the array of the database identifiers for the port instances retrieved.

Return

MWXSuccess	Occurs if the port instances are successfully retrieved.
MWXFail	Occurs if the port instances are not successfully retrieved. Note that if no port instances are attached to a cell instance, then zero objects are returned and it is considered a successful retrieval.

Example

```
rc = MWXDb_Get_PortInstFromCellInst (cellId,
cellInstId,&objectCount,&objectsIdArray);
switch (rc) {
    case MWXSuccess:
        if (objectCount > 0) {
            free (objSIdArray);
        }
        break;
    case MWXFail:
        MWX_Printf("ERROR -- Failed to get port
instances\n");
        break;
}
```

MWXDb_Get_PrBoundary

Retrieves the coordinates of the points defining the place and route boundary of a cell. The MWXDb_Get_CellBoundary function allocates memory to store the point array. It is your responsibility to recover this memory when you no longer need the data.

Syntax

```
int MWXDb_Get_PrBoundary(
    MWXCellId_t cellId, /* INPUT */
    int *numPoint, /* OUTPUT */
```

```

        MWXPoint_t **pointArray /* OUTPUT */
    )

```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell for which to retrieve the place and route boundary.
<code>numPoint</code>	Specifies the number of points in <code>pointArray</code> .
<code>pointArray</code>	Specifies the x/y-coordinate pairs representing the points that define the cell place and route boundary.

Return

<code>MWXSuccess</code>	Occurs if the place and route boundary coordinates are successfully retrieved.
<code>MWXFail</code>	Occurs if the place and route boundary coordinates are not successfully retrieved.

Example

```

rc = MWXDb_Get_PrBoundary (cellId, &numPoint, &pointArray);
    if (rc == MWXSuccess) {
        for (i = 0; i < numPoint; i++) {
            MWX_Printf("Cell point: (%d,%d)\n",
                pointArray[i].x, pointArray[i].y);
        }
        free (pointArray);
    }

```

MWXDb_Get_Property

Retrieves a property owned by a cell. The property can be attached to the cell or an object owned by the cell. The output parameters of the function are undefined if the function does not return `MWXSuccess`.

The property name returned from this procedure cannot be freed by the caller.

Note:

There is no need to free the memory for the `propertyName` field.

Syntax

```

int MWXDb_Get_Property (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t propertyId, /* INPUT */
    char **propertyName, /* OUTPUT */
    unsigned char *propertyDataType /* OUTPUT */
)

```

```

        MWXPropValue_t *propertyValue, /* OUTPUT */
        uint16 *propertySize /* OUTPUT */
    )

```

Parameters

cellId	Specifies the database identifier of the cell that owns the property.
propertyId	Specifies the database identifier of the property to retrieve.
propertyName	Specifies the name of the property retrieved.
propertyDataType	Specifies the type of data the property contains; valid values are defined in the mwX.h file and include MWXcPropBooleanTypeMWXcPropIntegerTypeMWXcPropFloatingTypeMWXcPropStringTypeMWXcPropDataStructTypeMWXcPropTimeType
propertyValue	Specifies the value of the property.
propertySize	Specifies the number of bytes the data requires for storage (in the case of a string, the size is the length of the string; in all other cases, it is the size of the data object).

Return

MWXSuccess	Occurs if the property is successfully retrieved.
MWXFail	Occurs if the property is not successfully retrieved.

Example

```

if (MWXDb_Get_Property (cellId, propertyId, &propertyName,&propertyDataType,
    &propertyValue,
                        &propertySize)
    == MWXSuccess) {
    switch (propertyDataType) {
        case MWXcPropBooleanType:
            ...
            break;
        case MWXcPropIntegerType:
            ...
            break;
        case MWXcPropFloatingType:
            ...
            break;
        case MWXcPropStringType:
            ...
            break;
        case MWXcPropDataStructType:
            ...
            break;
    }
}

```

```

        case MWXcPropTimeType:
            MWX_Printf("INFO -- found time property.\n");
            break;
        default:
            MWX_Printf("ERROR -- Unknown property type
                returned.\n");
    }
}

```

MWXDb_Get_Property_ByName

Retrieves a property attached to an object. The output parameters of the function are undefined if the function does not return MWXSuccess.

Syntax

```

int MWXDb_Get_Property_ByName (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t objectId, /* INPUT */
    char *propertyName, /* INPUT */
    MWXObjId_t *propertyId, /* OUTPUT */
    unsigned char *propertyDataType, /* OUTPUT */
    MWXPropValue_t *propertyValue, /* OUTPUT */
    uint16 *propertySize /* OUTPUT */
)

```

Parameters

cellId	Specifies the database identifier of cell that owns the object identified by <code>objectId</code> .
objectId	Specifies the database identifier of the object that owns the property.
propertyName	Specifies the name of the property to retrieve.
propertyId	Specifies the database identifier of the property retrieved.
propertyDataType	Specifies the type of data the property contains; valid values are defined in the <code>mwX.h</code> file and include <code>MWxcPropBooleanType</code> <code>MWxcPropIntegerType</code> <code>MWxcPropFloatingType</code> <code>MWxcPropStringType</code> <code>MWxcPropDataStructType</code> <code>MWxcPropTimeType</code>
propertyValue	Specifies the value of the property.
propertySize	Specifies the number of bytes the data requires for storage (in the case of a string, the size is the length of the string; in all other cases, it is the size of the data object).

Return

<code>MWXSucces</code>	Occurs if the property is successfully retrieved.
<code>MWXFail</code>	Occurs if the property is not successfully retrieved.

Example

```
if (MWXDb_Get_Property_ByName (cellId, objectId, propertyName,
    &propertyId,&propertyDataType, &propertyValue, &propertySize) == MWXSucces) {
    switch (propertyDataType) {
        case MWxcPropBooleanType:
            ...
            break;
        case MWxcPropIntegerType:
            ...
            break;
        case MWxcPropFloatingType:
            ...
            break;
        case MWxcPropStringType:
            ...
            break;
        case MWxcPropDataStructType:
            ...
            break;
        case MWxcPropTimeType:
            MWX_Printf("INFO -- found time property.\n");
```

`MWXDb_Get_Property_ByName`

```

        break;
default:
    MWX_Printf("ERROR -- Unknown property type
    returned.\n");
}}

```

MWXDb_Get_RefLib

Gets reference libraries for the specified design library. It is your responsibility to free the memory location after using the output of Reference Library Names.

Syntax

```

public int
MWXDb_Get_RefLib(
    char *libName, /* INPUT: library name */
    int32 *count, /* OUTPUT: count of reference lib */
    char ***refLibNames /* OUTPUT: reference library
        names array. It's users' responsibility to free
        the memory location after using.*/
)

```

Parameters

libName	Specifies the design library name.
count	Specifies the total number of reference libraries returned.
refLibNames	Specifies an array of all the reference library names in the design library.

Return

MWXSuccess	Occurs when the list of reference libraries is successfully retrieved.
MWXFail	Occurs when Milkyway fails to retrieve the list of reference libraries.

Example

```

void free_string_memory (MWXDBString_t *str) {
    if (str) {
        free(str);
    }
}

void Print_RefLib (char *libName) {
    int refCount = 0;

```



```

    int rc, i;
    char **refLibNames;

    rc = MWXDb_Get_RefLib(libName,&refCount,&refLibNames);
    if (rc != MWXSuccess) {
        MWX_Printf("FAIL - Unable to get Reference
                    libraries for library %s\n", libName);
        exit (MWXFail);
    }

    MWX_Printf("%d Reference Libraries for library %s are
                :\n",refCount, libName);
    for (i = 0; i < refCount; i++) {
        MWX_Printf("%s \n",refLibNames[i]);
    }
    free_char_memory(refLibNames);
}

```

MWXDb_Get_RefLibPathNames

Returns the number of reference libraries and their corresponding path names that were added to the *libName* library when you used the `dbAddRefLib` and `cmRefLib` commands (or Add > Ref in the GUI).

Note that reference libraries added through control files currently cannot be returned by `MWXDb_Get_RefLibPathNames`. To retrieve *all* reference libraries, use the `MWXDb_Get_RefLib` command. Be sure to free the memory that is allocated for `MWXDb_Get_RefLibPathNames` when the data is no longer needed.

Note:

The returned path name array contains the original path names of the reference libraries you entered when the reference library was added (unlike `MWXDb_Get_RefLib`, which always returns absolute paths).

Syntax

```

int MWXDb_Get_RefLibPathNames (
    char* libName, /* INPUT */
    int *count, /* OUTPUT */
    MWXDBString_t **refLibPathNames /* OUTPUT */
)

```

Parameters

libName	Specifies the path name to the library.
count	Returns a count of the number of reference library objects in the specified library.
refLibPathNames	Returns an array of strings that contain the path names of each reference library object in the specified library.

Return

MWXSuccess	Occurs if the path names are successfully retrieved.
MWXFail	Occurs if the path names are not successfully retrieved.

Dynamic Example

```
#include<stdio.h>
#include<stdlib.h>
#include "mwx.h"
#include "mwx_proto.h"

MWXSCM_t mwGetRefLibPathNames (MWXSCM_t list)
{
    char progDescrString[50] = "Get Reference Library relative path name -";
    char libName[256];
    int count, i;
    MWXDBString_t *refLibPathNames;
    static int caseSensitive = 0;
    /*-----
    GET THE SCHEME LIST
    -----*/
    if (MWXSc_Scanf (list, "s",libName) != 1) {
        MWX_Printf((char*)"Error.Synopsis : getRefLibPathNames libName\n");
        return (MWXFail);
    }
    /*-----
    START TEST FUNCTION
    -----*/
    MWX_Printf((char*)" \n%s Testing...\n", progDescrString);
    if (MWXDb_Get_RefLibPathNames(libName, &count, &refLibPathNames) != MWXSuccess)
    {
        MWX_Printf((char*) "Error : Can't get Reference Library relative path\n");
        return (MWXFail);
    }
    MWX_Printf((char*) "There are %d reference libraries in lib %s",count, libName);
    for (i=0; i< count; i++) {
        MWX_Printf("Reference Libname : %s\n",refLibPathNames[i]);
    }

    /* free allocated memory after used */
    if (count) {
```

```

    if (refLibPathNames) {
        free (refLibPathNames);
    }
}

/*-----*/
DONE
/*-----*/

MWX_Printf((char*)" %s DONE TESTING! \n\n",progDescrString);
return(MWXSuccess);
}
void reg_getRefLibPathNames(){
    MWXSc_Register_Function((char*)"getRefLibPathNames",mwGetRefLibPathNames);
}

```

Stand-Alone Example

```

int GetRefLibPathNames (char *libName)
{
    int count, i;
    MWXDBString_t *refLibPathNames;

    /*-----*/
    START TEST FUNCTION
    /*-----*/
    if (MWXDb_Get_RefLibPathNames(libName, &count, &refLibPathNames)
        != MWXSuccess)
    {
        MWX_Printf((char*) "Error : Can't get Reference Library relative path\n");
        return (MWXFail);
    }
    MWX_Printf((char*) "There are %d reference libraries in lib %s",count,
libName);
    for (i=0; i< count; i++) {
        MWX_Printf("Reference Libname : %s\n",refLibPathNames[i]);
    }

    /* free allocated memory after used */
    if (count) {
        if (refLibPathNames) {
            free (refLibPathNames);
        }
    }

    /*-----*/
    DONE
    /*-----*/

    return(MWXSuccess);
}

```

See Also

MWXDb_Get_RefLib

MWXDb_Get_RegionPort_ByName

This function retrieves regionPortId by given regionPortName.

Syntax

```
int MWXDb_Get_RegionPort_ByName(  
MWXCellId_t cellId, /* INPUT */  
char *regionPortName, /* INPUT */  
MWXObjId_t *regionPortId /* OUTPUT */  
)
```

Parameters

cellId	Specifies the cellId in the database
regionPortName	Specifies the region port name that you want to get the db identifier number
RegionPortId	Specifies the db identifier number of the regionPort

Return

MWXSuccess	Occurs if successfully gets the db ID of the regionPort
MWXFail	Occurs if fail to get the db ID of the region Port

Example

```
rc = MWXDb_Get_RegionPort_ByName(cellId, regionPortName, &regionPortId);  
switch (rc) {  
    case MWXSuccess:  
        printf("The regionPortId is %d\n", regionPortId);  
        break  
    case MWXFail:  
        printf("Fail to retrieve the regionPortId\n");  
        break;  
    Default:  
        printf("Unknown return code \n");  
        break;  
}
```

MWXDb_Get_RegionPort_Direction

This function retrieves the regionPort direction.

Syntax

```
int MWXDb_Get_CellInfo_cellNameAndView(  
MWXCellId_t cellId, /* INPUT */  
MWXObjId_t regionPortId /* INPUT */
```

```
unsigned char *direction /* OUTPUT */
)
```

Parameters

cellId	Specifies the cellId in the database
regionPortId	Specifies the ID of the regionPort that you want to get the direction from
direction	Direction of the region port

Return

MWXSuccess	Occurs if successfully gets regionPort's direction.
MWXFail	Occurs if fail to get the regionPort's direction

Example

```
rc = MWXDb_Get_RegionPort_Direction(cellId, regionPortId,&direction);
switch (rc) {
    case MWXSuccess:
        printf("The direction of the regionPort is %c\n", direction);
        break
    case MWXFail:
        printf("Fail to retrieve the cell and view name of the cell\n");
        break;
    Default:
        printf("Unknown return code \n");
        break;
}
```

MWXDb_Get_SelectedSet

Retrieves database identifiers for all the selected objects in a cell in a given graphics window. It is your responsibility to recover the memory that is allocated for the `objectsIdArray` field when the data is no longer needed.

Syntax

```
int MWXDb_Get_SelectedSet (
    int windowNumber,           /* INPUT */
    int *objectCount,           /* OUTPUT */
    MWXObjId_t **objectsIdArray /* OUTPUT */
)
```

Parameters

windowNumber	The number of the graphics window that contains the selected objects.
objectCount	Specifies the number of selected objects.
objectsIdArray	Specifies the array of the database identifiers for the selected objects retrieved.

Return

MWXSuccess	Occurs if the database identifiers of the selected objects are successfully retrieved.
MWXFail	Occurs if the database identifiers of the selected objects are not successfully retrieved.

Example

```
rc = MWXDb_Get_SelectedSet (windowNumber, &objectCount,
    &objectsIdArray);

switch (rc) {
    case MWXSuccess:
        for (i = 0; i < objectCount; i++)
            MWX_Printf("%d\n", objectsIdArray[i]);
        if (objectCount > 0)
            free (objectsIdArray);
        return (MWXSuccess);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to get selected objects.\n");
        return (MWXFail);
    default:
        MWX_Printf("ERROR -- Unknown error occurred. \n");
        return (MWXFail);
}
```

Scheme Equivalent

geGetSelectedSet
(see Physical Implementation Online Help)

Note:

This function is applicable only to Astro, the Jupiter tool, and Milkyway.

MWXdB_Get_ViaRegionFromPort

Retrieves the ViaRegion record from a port. It is your responsibility to free the allocated memory for the `objectsIdArray` field when the data is no longer needed.

Syntax

```
int MWXdB_Get_ViaRegionFromPort (
    MWXCellId_t cellId, /*INPUT*/
    MWXObjId_t portId, /*INPUT: only get the object
        IDs attached to this portid */
    int32 *objectCount, /*OUTPUT: number of members */
    MWXObjId_t **objectsIdArray /*OUTPUT: an array of
        all member id. It is the caller's responsibility
        to free the memory, i.e., free (*memberIdArray). */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of a cell.
<code>portId</code>	Specifies the database identifier of the port.
<code>objectCount</code>	Specifies the number of the objects being returned in <code>objectsIdArray</code> .
<code>objectsIdArray</code>	Specifies the array of database identifiers for objects retrieved.

Return

<code>MWXSucces</code>	Occurs if the record is successfully retrieved.
<code>MWXFail</code>	Occurs if the record is not successfully retrieved.

Example

```
rc = MWXdB_Get_ViaRegionFromPort (cellId, portId, &objectCount,
&objectsIdArray);
    MWX_Printf("INFO -- rc=%d\n",rc);
    switch (rc) {
        case MWXSucces:
            for (i = 0; i < objectCount; i++) {
                sprintf(buf, "(define objects (append objects '(%d)))",
objectsIdArray[i]);
                MWXSc_Evaluate (buf);
            }

            if (objectCount > 0) free (objectsIdArray);
            return (MWXSc_Make_IntegerSCM (objectCount));
        case MWXFail:
            MWX_Printf("ERROR -- Failed to get viaRegion IDs\n");
    }
```

```

        return (MWXFail);
    default:
        MWX_Printf("ERROR -- unknown return code from
MWXDb_Get_ViaRegionFromPort\n");
        return (MWXFail);
}

```

MWXDb_Get_WireDirection_layer

Retrieves the layer for a given wire direction object.

Syntax

```

int MWXDb_Get_WireDirection_layer(
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t objId, /* INPUT */
    int32 *value /* OUTPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell.
objId	Specifies the database identifier of the wire track object.
value	Specifies the returned value of the layer.

Return

MWXSuccess	Occurs when the layer is successfully retrieved.
MWXFail	Occurs when the layer retrieval fails.

Example

```

rc = MWXDb_Get_WireDirection_layer(cellId, objId, &value);
switch (rc) {
    case MWXSuccess:
        MWX_Printf("The returned field layer: %d \n",
            value);
    case MWXFail:
        MWX_Printf("Fail to get the desired field.
            \n");
        return (MWXFail)
    default:
        return (MWXFail);
}

```

MWXDb_Get_WireDirection_preferedDirection

Retrieves the preferred direction for a given wire direction object.

Syntax

```
int MWXDb_Get_WireDirection_preferedDirection(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t objId, /* INPUT */  
    int32 *value /* OUTPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell.
objId	Specifies the database identifier of the wire track object.
value	Specifies the returned value of the preferred direction.

Return

MWXSuccess	Occurs when the preferred direction is successfully retrieved.
MWXFail	Occurs when the preferred direction retrieval fails.

Example

```
rc = MWXDb_Get_WireDirection_preferedDirection(cellId, objId, &value);  
    switch (rc) {  
        case MWXSuccess:  
            MWX_Printf("The returned field preferred  
                direction: %d \n", value);  
            return (MWXSuccess);  
        case MWXFail:  
            MWX_Printf("Fail to get the desired field.  
                \n");  
            return (MWXFail);  
        default:  
            return (MWXFail);  
    }
```

MWXDb_Get_WireMasterId

Retrieves the wireMasterId from the database.

Syntax

```
int MWXDb_Get_WireMasterId (
    MWXCellId_t cellId, /*INPUT*/
    uchar layer, /*INPUT*/
    uchar wireMasterIdx, /*INPUT*/
    MWXObjId_t *wireMasterId /* OUTPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that owns the wireMasterId.
layer	Specifies the layer number.
wireMasterIdx	The index for searching the WireMaster identifier.
wireMasterId	Specifies the database identifier of the WireMaster.

Return

MWXSuccess	Occurs if the function successfully retrieves the wireMasterId.
MWXFail	Occurs if the function fails to successfully retrieve the wireMasterId.

Example

```
rc = MWXDb_Get_WireMasterId( cellId, layer, wireMasterIdx, &wireMasterId );
switch (rc) {
    case MWXSuccess:
        return (MWXSc_Make_IntegerSCM (wireMasterId));
    case MWXFail:
        MWX_Printf("ERROR -- Failed to retrieve the
        wireMasterId");
        return (MWXFail);
}
```

MWXDb_Get_WireTrack_bbox

Retrieves the bbox for a given wire track object.

Syntax

```
int MWXDb_Get_WireTrack_bbox(
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t objId, /* INPUT */
    int32 *left, /* OUTPUT */
    int32 *bottom, /* OUTPUT */
    int32 *right, /* OUTPUT */
)
```

```

        int32 *top /* OUTPUT */
    )

```

Parameters

cellId	Specifies the database identifier of the cell.
objId	Specifies the database identifier of the wire track object.
left	Specifies the x-coordinate of the left side of the bounding box.
bottom	Specifies the y-coordinate of the bottom of the bounding box.
right	Specifies the x-coordinate of the right side of the bounding box.
top	Specifies the y-coordinate of the top of the bounding box.

Return

MWXSucces	Occurs when the bbox is successfully retrieved.
MWXFail	Occurs when the bbox retrieval fails.

Example

```

rc = MWXDb_Get_WireTrack_bbox(cellId, objId, &left, &bottom, &right, &top);
switch (rc) {
    case MWXSucces:
        MWX_Printf("The bbox :  (%d, %d, %d, %d) \n",
            left, bottom, right, top);
        return (MWXSucces);
    case MWXFail:
        MWX_Printf("Fail to get the desired field.
            \n");
        return (MWXFail);
    default:
        return (MWXFail);
}

```

MWXDb_Get_WireTrack_layer

Retrieves the layer for a given wire track object.

Syntax

```

int MWXDb_Get_WireTrack_layer(
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t  objId, /* INPUT */
    int32 *value /* OUTPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell.
objId	Specifies the database identifier of the wire track object.
value	Specifies the returned value of the layer.

Return

MWXSuccess	Occurs when the layer is successfully retrieved.
MWXFail	Occurs when the layer retrieval fails.

Example

```
rc = MWXDb_Get_WireTrack_layer(cellId, objId, &value);
    switch (rc) {
        case MWXSuccess:
            MWX_Printf("The returned field layer: %d \n",
                value);
            return (MWXSuccess);
        case MWXFail:
            MWX_Printf("Fail to get the desired field.
                \n");
            return (MWXFail);
        default:
            return (MWXFail);
    }
```

MWXDb_Get_WireTrack_startTrack

Retrieves the start track direction for a given wire track object.

Syntax

```
int MWXDb_Get_WireTrack_startTrack(
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t objId, /* INPUT */
    int32 *value /* OUTPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell.
objId	Specifies the database identifier of the wire track object.
value	Specifies the returned value of the start track direction.

Return

MWXSuccess	Occurs when the start track is successfully retrieved.
MWXFail	Occurs when the start track retrieval fails.

Example

```
rc = MWXDb_Get_WireTrack_startTrack(cellId, objId, &value);
    switch (rc) {
        case MWXSuccess:
            MWX_Printf("The returned field start track: %d\n", value);
            return (MWXSuccess);
        case MWXFail:
            MWX_Printf("Fail to get the desired field.\n");
            return (MWXFail);
        default:
            return (MWXFail);
    }
```

MWXDb_Get_WireTrack_times

Retrieves the times for a given wire track object.

Syntax

```
int MWXDb_Get_WireTrack_times(
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t objId, /* INPUT */
    int32 *value /* OUTPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell.
objId	Specifies the database identifier of the wire track object.
value	Specifies the returned value of the times.

Return

MWXSuccess	Occurs when the times are successfully retrieved.
MWXFail	Occurs when the times retrieval fails.

Example

```
rc = MWXDb_Get_WireTrack_times(cellId, objId, &value);
    switch (rc) {
        case MWXSuccess:
            MWX_Printf("The returned field times: %d \n",
                value);
            return (MWXSuccess);
        case MWXFail:
            MWX_Printf("Fail to get the desired field.
                \n");
            return (MWXFail);
        default:
            return (MWXFail);
    }
```

MWXDb_Get_WireTrack_trackDirection

Retrieves the track direction for a given wire track object.

Syntax

```
int MWXDb_Get_WireTrack_trackDirection(
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t objId, /* INPUT */
    int32 *value /* OUTPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell.
objId	Specifies the database identifier of the wire track object.
value	Specifies the returned value of the track direction.

Return

MWXSuccess	Occurs when the track direction is successfully retrieved.
MWXFail	Occurs when the track direction retrieval fails.

Example

```
rc = MWXDb_Get_WireTrack_trackDirection(cellId,
objId, &value);
    switch (rc) {
        case MWXSuccess:
            MWX_Printf("The returned field Track
                Direction: %d \n", value);
            return (MWXSuccess);
        case MWXFail:
            MWX_Printf("Fail to get the desired field.
                \n");
            return (MWXFail);
        default:
            return (MWXFail);
    }
```

MWXDb_Get_WireTrack_trackStep

Retrieves the track step for a given wire track object.

Syntax

```
int MWXDb_Get_WireTrack_trackStep(
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t objId, /* INPUT */
    int32 *value /* OUTPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell.
objId	Specifies the database identifier of the wire track object.
value	Specifies the returned value of the track step.

Return

MWXSuccess	Occurs when the track step is successfully retrieved.
MWXFail	Occurs when the track step retrieval fails.

Example

```
rc = MWXDb_Get_WireTrack_trackStep(cellId, objId, &value);
    switch (rc) {
        case MWXSuccess:
            MWX_Printf("The returned field track step : %d\n", value);
            return (MWXSuccess);
        case MWXFail:
            MWX_Printf("Fail to get the desired field.\n");
            return (MWXFail);
        default:
            return (MWXFail);
    }
```

MWXDb_Get_WireTrack_type

Retrieves the type for a given wire track object.

Syntax

```
int MWXDb_Get_WireTrack_type(
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t objId, /* INPUT */
    int32 *value /* OUTPUT */
)
```


Parameters

cellId	Specifies the database identifier of the cell.
objId	Specifies the database identifier of the wire track object.
value	Specifies the returned value of the type.

Return

MWXSuccess	Occurs when the type is successfully retrieved.
MWXFail	Occurs when the type retrieval fails.

Example

```
rc = MWXDb_Get_WireTrack_type(cellId, objId, &value);
switch (rc) {
    case MWXSuccess:
        MWX_Printf("The returned field type: %d \n",
            value);
        return (MWXSuccess);
    case MWXFail:
        MWX_Printf("Fail to get the desired field.
            \n");
        return (MWXFail);
    default:
        return (MWXFail);
}
```

MWXDb_Is_CaseSensitive

Returns the case sensitivity of the given library.

Syntax

```
int MWXDb_Is_CaseSensitive (
    MWXLibId_t libId /* INPUT */
)
```

Parameters

`libId` Specifies the ID of the library to be queried.

Return

1 Occurs if the library is case sensitive.

0 Occurs if the library is not case sensitive.

Dynamic Example

```
MWXSCM_t mwIsCaseSensitive(MWXSCM_t list) {
    MWXLibId_t libId;
    int caseSensitive;

    if (MWXSc_Scanf(list, "o", &libId) != 1) {
        MWX_Printf("ERROR -- mwIsCaseSensitive requires 1 argument.\n");
        return MWX_FALSE_SCM;
    }

    caseSensitive = MWXDb_Is_CaseSensitive(libId);

    if (caseSensitive)
        MWX_Printf("Library is case sensitive\n");
    else
        MWX_Printf("Library is not case sensitive\n");

    return MWX_TRUE_SCM;
}
```

Stand-Alone Example

```
int IsCaseSensitive(MWXLibId_t libId) {

    int caseSensitive;

    MWX_Printf("INFO -- MWXDb_Is_CaseSensitive.\n");
    caseSensitive = MWXDb_Is_CaseSensitive(libId);

    if (caseSensitive)
        MWX_Printf("Library is case sensitive\n");
    else
        MWX_Printf("Library is not case sensitive\n");

    return MWXSuccess;
}
```

Scheme Equivalent

`dbIsCaseSensitive`

MWXdB_Is_NetSubTypeCoreAndPad

Checks to see if the netSubType is the coreAndpad subtype or not.

Syntax

```
int MWXdB_Is_NetSubTypeCoreAndPad(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t netId /* INPUT */  
)
```

Parameters

cellId	Database object ID of the cell to which the information applies.
netId	Specifies the database identifier of the net to which the information applies.

Return

MWXSuccess	Occurs if the netSubType is CoreAndPad.
MWXFail	Occurs if the netSubType is not CoreAndPad.

Example

```
if (MWXdB_Is_NetSubTypeCoreAndPad(cellId,netId) != MWXSuccess) {  
    MWX_Printf("NetSubType is not CoreAndPad\n");  
    return (MWXFail);  
}  
return (MWXSuccess);
```

MWXdB_Is_NetSubTypeCoreOnly

Checks to see if the netSubType is the coreOnly subtype or not.

Syntax

```
int MWXdB_Is_NetSubTypeCoreOnly(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t netId /* INPUT */  
)
```

Parameters

cellId	Database object ID of the cell to which the information applies.
NetId	Specifies the database identifier of the net to which the information applies

Return

MWXSucces	Occurs if the netSubType is coreOnly.
MWXFail	Occurs if the netSubType is not coreOnly.

Example

```
if (MWXDb_Is_NetSubTypeCoreOnly(cellId,netId) != MWXSucces) {  
    MWX_Printf("NetSubType is not Core Only\n");  
    return (MWXFail);  
}  
return (MWXSucces);
```

MWXDb_Is_NetSubTypeCTSNet

Checks to see if the netSubType is the CTSNet subtype or not.

Syntax

```
int32 MWXDb_Is_NetSubTypeCTSNet(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t netId /* INPUT */  
)
```

Parameters

cellId	Database object ID of the cell to which the information applies.
NetId	Specifies the database identifier of the net to which the information applies.

Return

MWXSuccess	Occurs if the netSubType is CTSNet.
MWXFail	Occurs if the netSubType is not CTSNet.

Example

```
if (MWXDb_Is_NetSubTypeCTSNet (cellId,netId) != MWXSuccess) {  
    MWX_Printf("NetSubType is not CTSNet\n");  
    return (MWXFail);  
}  
return (MWXSuccess);
```

MWXDb_Is_NetSubTypeCTSRootNet

Checks to see if the netSubType is the CTSRootNet subtype or not.

Syntax

```
int32 MWXDb_Is_NetSubTypeCTSRootNet(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t netId /* INPUT */  
)
```

Parameters

<code>cellId</code>	Database object ID of the cell to which the information applies.
<code>NetId</code>	Specifies the database identifier of the net to which the information applies.

Return

<code>MWXSuccess</code>	Occurs if the <code>netSubType</code> is <code>CTSRotNet</code> .
<code>MWXFail</code>	Occurs if the <code>netSubType</code> is not <code>CTSRotNet</code> .

Example

```
if (MWXDb_Is_NetSubTypeCTSRotNet (cellId,netId) != MWXSuccess) {  
    MWX_Printf("NetSubType is not CTSRotNet\n");  
    return (MWXFail);  
}  
return (MWXSuccess);
```

MWXDb_Is_NetSubTypeNonCTSNet

Checks to see if the `netSubType` is the `NonCTSNet` subtype or not.

Syntax

```
int32 MWXDb_Is_NetSubTypeNonCTSNet(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t netId /* INPUT */  
)
```

Parameters

cellId	Database object ID of the cell to which the information applies.
NetId	Specifies the database identifier of the net to which the information applies.

Return

MWXSucces	Occurs if the netSubType is NonCTSNet.
MWXFail	Occurs if the netSubType is not NonCTSNet.

Example

```
if (MWXDb_Is_NetSubTypeNonCTSNet (cellId,netId) != MWXSucces) {  
    MWX_Printf("NetSubType is not NonCTSNet\n");  
    return (MWXFail);  
}  
return (MWXSucces);
```

MWXDb_Is_NetSubTypePadOnly

Checks to see if the netSubType is the padOnly subtype or not.

Syntax

```
int MWXDb_Is_NetSubTypePadOnly(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t netId /* INPUT */  
)
```

Parameters

<code>cellId</code>	Database object ID of the cell to which the information applies.
<code>netId</code>	Specifies the database identifier of the net to which the information applies.

Return

<code>MWXSuccess</code>	Occurs if the <code>netSubType</code> is <code>padOnly</code> .
<code>MWXFail</code>	Occurs if the <code>netSubType</code> is not <code>padOnly</code> .

Example

```
if (MWXDb_Is_NetSubTypePadOnly(cellId,netId) != MWXSuccess) {  
    MWX_Printf("NetSubType is not Pad Only\n");  
    return (MWXFail);  
}  
return (MWXSuccess);
```

MWXDb_Is_Object_InGroup

Tests for the existence of an object in a specified group.

Syntax

```
int MWXDb_Is_Object_InGroup (  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t groupId, /* INPUT */  
    MWXObjId_t objectId /* INPUT */  
)
```


Parameters

cellId	Specifies the database identifier of the cell that owns the group and the object.
groupId	Specifies the database identifier of the group within which the object is searched for.
objectId	Specifies the database identifier of the object.

Return

MWXNoLicense	Occurs if the Synopsys license is not present.
MWXFail	Occurs if the object is not successfully found.
MWXSuccess	Occurs if the object is in the group

Example

```
rc = MWXDb_Is_Object_InGroup( cellId, groupId, objectId );
switch (rc) {
    case MWXSuccess:
        return (MWXSuccess);
    case MWXFail:
        return (MWXFail);
}
```

MWXDb_Iter_{Object}To{Attachment}

Starts the iteration to retrieve the database identifier of an object that is attached to the object specified by the object ID (objId). This command returns an iterator that allow you to get a list of attachment objects. It is your responsibility to release the iterator.

Use `MWXDb_Iter_Next` to get next attachment object.

Use `MWXDb_Iter_Quit` to terminate attachment object iteration.

Syntax

```
int MWXDb_Iter_{Object}To{Attachment} (
    MWXCellId_t cellId,          /* INPUT */
    MWXObjId_t objId,           /* INPUT */
    MWXIteratorId_t *iterId     /* OUTPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell containing the object.
objId	Specifies the database identifier of the object from which the attached object will be retrieved.
iterId	Specifies the iterator identifier.

Return

MWXSucces	Occurs if the iterator was successfully returned.
MWXFail	Occurs if the iterator was not successfully returned.

Prototypes

```
/* For ExtraPort, ExtraPin, ExtraPortInstMaster, and ExtraPinInstMaster */
int MWXDb_Iter_ExtraPortToExtraPin (MWXCellId_t cellId, MWXObjId_t objId,
    MWXIteratorId_t *iterId);

int MWXDb_Iter_ExtraPortToViaRegion (MWXCellId_t cellId, MWXObjId_t objId,
    MWXIteratorId_t *iterId);

int MWXDb_Iter_ExtraPortToTiming (MWXCellId_t cellId, MWXObjId_t objId,
    MWXIteratorId_t *iterId);

int MWXDb_Iter_ExtraPortInstMasterToExtraPinInstMaster (MWXCellId_t cellId,
    MWXObjId_t objId, MWXIteratorId_t *iterId);

/* For Hierarchical Objects */
int MWXDb_Iter_HierCellInstMasterToHierPortInstMaster P_((MWXCellId_t cellId,
    MWXObjId_t objId, MWXIteratorId_t *iterId));

int MWXDb_Iter_HierCellInstToHierPortInst P_((MWXCellId_t cellId,
    MWXObjId_t objId, MWXIteratorId_t *iterId));

int MWXDb_Iter_HierCellInstToChildHierCellInst P_((MWXCellId_t cellId,
    MWXObjId_t objId, MWXIteratorId_t *iterId));

int MWXDb_Iter_HierCellInstToChildCellInst P_((MWXCellId_t cellId,
    MWXObjId_t objId, MWXIteratorId_t *iterId));

int MWXDb_Iter_HierNetToHierPortInst P_((MWXCellId_t cellId,
    MWXObjId_t objId, MWXIteratorId_t *iterId));

/* For Other Objects */
int MWXDb_Iter_CellInstToPortInst P_((MWXCellId_t cellId, MWXObjId_t objId,
    MWXIteratorId_t *iterId));
int MWXDb_Iter_CellInstMasterToCellInst P_((MWXCellId_t cellId,
    MWXObjId_t objId,
    MWXIteratorId_t *iterId));
int MWXDb_Iter_CellInstMasterToPortInstMaster P_((MWXCellId_t cellId,
```

```

    MWXObjId_t objId, MWXIteratorId_t *iterId));
int MWXDb_Iter_CellRegionToCellInst P_((MWXCellId_t cellId, MWXObjId_t objId,
    MWXIteratorId_t *iterId));
int MWXDb_Iter_CellRegionToCellRow P_((MWXCellId_t cellId, MWXObjId_t objId,
    MWXIteratorId_t *iterId));
int MWXDb_Iter_CellRowToCellInst P_((MWXCellId_t cellId, MWXObjId_t objId,
    MWXIteratorId_t *iterId));

int MWXDb_Iter_NetToPort P_((MWXCellId_t cellId, MWXObjId_t objId,
    MWXIteratorId_t *iterId));
int MWXDb_Iter_NetToPortInst P_((MWXCellId_t cellId, MWXObjId_t objId,
    MWXIteratorId_t *iterId));
int MWXDb_Iter_NetToWiring P_((MWXCellId_t cellId, MWXObjId_t objId,
    MWXIteratorId_t *iterId));

int MWXDb_Iter_PinToAccessEdge P_((MWXCellId_t cellId, MWXObjId_t objId,
    MWXIteratorId_t *iterId));
int MWXDb_Iter_PortToPin P_((MWXCellId_t cellId, MWXObjId_t objId,
    MWXIteratorId_t *iterId));
int MWXDb_Iter_PortInstMasterToPinInstMaster P_((MWXCellId_t cellId,
    MWXObjId_t objId, MWXIteratorId_t *iterId));

int MWXDb_Iter_RLCNetToPort P_((MWXCellId_t cellId, MWXObjId_t objId,
    MWXIteratorId_t *iterId));
int MWXDb_Iter_RLCNetToNode P_((MWXCellId_t cellId, MWXObjId_t objId,
    MWXIteratorId_t *iterId));
int MWXDb_Iter_RLCNetToREdge P_((MWXCellId_t cellId, MWXObjId_t objId,
    MWXIteratorId_t *iterId));
int MWXDb_Iter_RLCNetToLEdge P_((MWXCellId_t cellId, MWXObjId_t objId,
    MWXIteratorId_t *iterId));
int MWXDb_Iter_RLCNetToCoupleNet P_((MWXCellId_t cellId, MWXObjId_t objId,
    MWXIteratorId_t *iterId));
int MWXDb_Iter_RLCCoupleNetToCEdge P_((MWXCellId_t cellId, MWXObjId_t objId,
    MWXIteratorId_t *iterId));
int MWXDb_Iter_RLCCoupleNetToLEdge P_((MWXCellId_t cellId, MWXObjId_t objId,
    MWXIteratorId_t *iterId));

```

Example

```

MWXCellId_t      cellId;
MWXObjId_t       netId, portInstId, wiringId;
MWXIteratorId_t  netIterId, portInstIterId, wiringIterId;
MWXObjTypeId_t   objType;
MWXPoint_t       *point;
char             *netName;
int              rc, rc2;

if (MWXDb_Iter_ObjectByType( cellId, MWX_NET, &netIterId)
    != MWXSuccess) {
    MWX_Printf((char *) "Failed to iterate net object\n");
    return (MWXFail);
}

while ((rc = MWXDb_Iter_Next( netIterId, &netId))
    == MWXSuccess) {
    MWXDb_Get_Net_name( cellId, netId, &netName);
    MWX_Printf((char *) "Net name = %s\n", netName);
}

```

```

/* Port instance attachment */
MWXDb_Iter_NetToPortInst( cellId, netId, &portInstIterId);
while ((rc2 = MWXDb_Iter_Next( portInstIterId, &portInstId))
      == MWXSuccess) {
    MWX_Printf((char *)"Port instance Id : %d\n", portInstId);
}
MWXDb_Iter_Quit( portInstIterId);

/* Wiring attachment */
MWXDb_Iter_NetToWiring( cellId, netId, &wiringIterId);
while ((rc2 = MWXDb_Iter_Next( wiringIterId, &wiringId))
      == MWXSuccess) {
    MWXDb_Get_ObjectsType( cellId, wiringId, &objType);
    switch(objType) {
    case MWX_VERTICALWIRE:
        MWXDb_Get_VerticalWire_startPoint( cellId, wiringId, &point);
        MWX_Printf((char *)" Vertical wire (id=%d objType=%d) : start point (%d, %d)\n",
                    wiringId, objType, point->x, point->y);
        break;
    case MWX_HORIZONTALWIRE:
        MWXDb_Get_HorizontalWire_startPoint( cellId, wiringId, &point);
        MWX_Printf((char *)" Horizontal wire (id=%d objType=%d) : start point (%d, %d)\n",
                    wiringId, objType, point->x, point->y);
        break;
    case MWX_CONTACT:
        MWXDb_Get_Contact_centerPosition( cellId, wiringId, &point);
        MWX_Printf((char *)" Contact (id=%d objType=%d) : center point (%d, %d)\n",
                    wiringId, objType, point->x, point->y);
        break;
    default:
        break;
    }
}
MWXDb_Iter_Quit( wiringIterId);
}

if (rc == MWXIterDone)
    MWX_Printf((char *)"Done with net object iteration.\n");
else
    MWX_Printf((char *)"Failed during the iteration of net object\n");

MWXDb_Iter_Quit( netIterId);

```

See Also

[MWXDb_Iter_Next](#)

[MWXDb_Iter_Quit](#)

MWXdB_Iter_LibCellViewPair

Creates an iterator for a currently open library (the design library or any of its reference libraries) and returns a handle to the iterator. Subsequent calls to `MWXdB_Iter_Next_LibCellViewPair` with the handle will iterate through all the cells in every view of the library. If the mode is not equal to 0, the iteration continues to include all the library's reference libraries as well.

The number of versions for each cell in a view does not affect the iteration, as each cell-view pair is accounted for once. For example, even if "top_design:1", "top_design:2", and "top_design:3" exist inside the CEL view, there will only be one occurrence of the cell-view pair "top_design"-"CEL".

Syntax

```
int32 MWXdB_Iter_LibCellViewPair(  
    MWXLibId_t libId, /* INPUT */  
    int32 mode, /* INPUT */  
    MWXIteratorId_t *iterId /* OUTPUT */  
)
```

Parameters

<code>libId</code>	Specifies the database ID of the library to iterate.
<code>mode</code>	Determines which libraries are included in the iteration: mode == 0, indicates that the iteration includes only the specified library. mode != 0, indicates that the iteration includes the specified library and all its reference libraries.
<code>iterId</code>	Returns a handle to the iterator.

Returns

<code>MWXSuccess</code>	Occurs if the iterator is created successfully and <code>iterId</code> is returned.
<code>MWXFail</code>	Occurs if the iterator is not created successfully.

Dynamic Example

See example in [MWXdB_Iter_Next_LibCellViewPair](#)

Stand-Alone Example

See example in [MWXdB_Iter_Next_LibCellViewPair](#)

See Also

[MWXDb_Iter_Next_LibCellViewPair](#)

[MWXDb_Iter_Quit_LibCellViewPair](#)

MWXDb_Iter_Next

Retrieves the next database identifier with the given iterator.

Syntax

```
int MWXDb_Iter_Next (
    MWXIteratorId_t iterId, /* INPUT */
    MWXObjId_t *objId, /* OUTPUT */
)
```

Parameters

<code>iterId</code>	Specifies the iterator identifier.
<code>objId</code>	Specifies the database identifier for the object to be retrieved.

Return

<code>MWXSuccess</code>	Occurs if the iterator is successfully returned.
<code>MWXIterDone</code>	Occurs if the end of object iteration is reached.
<code>MWXFail</code>	Occurs if the iterator is not successfully returned.

Example

```
int queryNetName (
    MWXCellId_t    cellId
    MWXObjId_t     netId
)
{
    MWXIteratorId_t netIterId;
    char            *netName;
    int             rc;

    if (MWXDb_Iter_ObjectByType( cellId, MWX_NET, &netIterId) != MWXSuccess) {
        MWX_Printf((char *) "Failed to iterate net object\n");
        return 0;
    }

    while ((rc = MWXDb_Iter_Next( netIterId, &netId)) == MWXSuccess) {
        if (MWXDb_Get_Net_name( cellId, netId, &netName) != MWXSuccess) {
            MWX_Printf((char*) "Fail to get net name\n");
        }
    }
}
```

```

        MWX_Printf ((char *) "Net name = %s\n", netName);
    }

    if (rc == MWXIterDone)
        MWX_Printf((char *) "Done with net object iteration.\n");
    else
        MWX_Printf((char *) "Failed during the iteration of net object\n");

    MWXDb_Iter_Quit( netIterId);
    return 1;
}

```

See Also

[MWXDb_Iter_ObjectByType](#)
[MWXDb_Iter_Quit](#)

MWXDb_Iter_Next_LibCellViewPair

When passed `iterId`, this function returns the next cell-view name pair of the iterator or `MWXIterDone` if the iteration is complete. You need to call `MWXDb_Iter_Quit_LibCellViewPair` to actually quit the iterator.

Syntax

```

int32 MWXDb_Iter_LibCellViewPair(
    MWXIteratorId_t *iterId /* INPUT/OUTPUT */
    char **cellName, /* OUTPUT */
    char **viewName /* OUTPUT */
)

```

Parameters

<code>iterId</code>	Specifies which iterator to use and returns a handle to the next element in the iteration.
<code>cellName</code>	Returns the name of the cell the iterator is currently on.
<code>viewName</code>	Returns the name of the view the iterator is currently on.

Returns

<code>MWXSucces</code>	Occurs if a new cell-view pair is successfully returned.
<code>MWXIterDone</code>	Occurs if the iterator has finished.
<code>MWXFail</code>	Occurs if the iterator fails to return a cell-view pair for any other reason.

Dynamic Example

```
/* Iterates through all the cell-view pairs in the given library (plus
   its reference libraries if mode != 0) and prints them to the screen */

MWXSCM_t mwIterateLibCellViewPair(MWXSCM_t list) {

    MWXIteratorId_t iterId;
    char libName[256];
    char *cellName;
    char *viewName;
    MWXLibId_t libId;
    int32 mode;
    int rc;

    if (MWXSc_Scanf(list, "sd", libName, &mode) != 2) {
        MWX_Printf("ERROR -- mwIterateLibCellViewPair requires 2 arguments.\n");
        return (MWXFail);
    }

    MWXDb_Open_Lib(libName, &libId);

    MWX_Printf("INFO -- MWXDb_Iter_LibCellViewPair.  Creating iterator for %s with
               mode %d\n", libName, mode);
    rc = MWXDb_Iter_LibCellViewPair(libId, mode, &iterId);

    if (rc == MWXSucces) {

        MWX_Printf("INFO -- MWXDb_Iter_Next_LibCellViewPair.  Retrieving cell-view
                   name pairs.\n");
        while ((rc = MWXDb_Iter_Next_LibCellViewPair(&iterId, &cellName,
                                                    &viewName)) == MWXSucces) {
            MWX_Printf("%s.%s\n", cellName, viewName);
        }
    }
}
```



```

if (rc != MWXIterDone) {
MWX_Printf("ERROR -- Unable to retrieve next cell-view name pair.
           MWXDb_Iter_Next_LibCellViewPair failed.\n");
return (MWXFail);
}

MWX_Printf("INFO -- MWXDb_Iter_Quit_LibCellViewPair. Closing iterator.\n");
if ((rc = MWXDb_Iter_Quit_LibCellViewPair (iterId)) == MWXSuccess) {
MWX_Printf("Successfully closed iterator.\n");
return (MWXSuccess);
} else {
MWX_Printf("ERROR -- Unable to close the iterator.
           MWXDb_Iter_Quit_LibCellViewPair failed.\n");
return (MWXFail);
}

} else {
MWX_Printf("ERROR -- Unable to create iterator.  MWXDb_Iter_LibCellViewPair
           failed.\n", libName, mode);
return (MWXFail);
}
}

```

Stand-Alone Example

```

int IterateLibCellViewPair(char *libName, int32 mode) {

int rc;
MWXLibId_t libId;
MWXIteratorId_t iterId;
char *cellName;
char *viewName;

MWXDb_Open_Lib(libName, &libId);
MWX_Printf("INFO -- MWXDb_Iter_LibCellViewPair.  Creating iterator for %s with
           mode %d\n", libName, mode);
rc = MWXDb_Iter_LibCellViewPair(libId, mode, &iterId);

if (rc == MWXSuccess) {

MWX_Printf("INFO -- MWXDb_Iter_Next_LibCellViewPair.  Retrieving cell-view
           name pairs.\n");
while ((rc = MWXDb_Iter_Next_LibCellViewPair(&iterId, &cellName,
           &viewName))== MWXSuccess) {
    MWX_Printf("%s.%s\n", cellName, viewName);
}

if (rc != MWXIterDone) {
MWX_Printf("ERROR -- Unable to retrieve next cell-view name pair.
           MWXDb_Iter_Next_LibCellViewPair failed.\n");
return (MWXFail);
}

MWX_Printf("INFO -- MWXDb_Iter_Quit_LibCellViewPair.  Closing
           iterator.\n");
if (MWXDb_Iter_Quit_LibCellViewPair (iterId) == MWXSuccess) {
    MWX_Printf("Successfully closed iterator.\n");
    return (MWXSuccess);
} else {

```

```

MWX_Printf("ERROR -- Unable to close the iterator.
           MWXDb_Iter_Quit_LibCellViewPair failed.\n");
return (MWXFail);
}

} else {
MWX_Printf("ERROR -- Unable to create itereator.  MWXDb_Iter_LibCellViewPair
           failed.\n", libName, mode);
return (MWXFail);
}
}

```

See Also

[MWXDb_Iter_LibCellViewPair](#)

[MWXDb_Iter_Quit_LibCellViewPair](#)

MWXDb_Iter_ObjectByType

Starts the iteration to retrieve the database identifier for the object type (`objType`) that is contained in the cell. This function returns an iterator that enables you to get the next object. The iteration continues until you terminate it.

Note:

Use the `MWXDb_Iter_Next` command to get the next object. Use the `MWXDb_Iter_Quit` command to terminate the object iteration.

Syntax

```

int MWXDb_Iter_ObjectByType (
    MWXCellId_t cellId, /* INPUT */
    MWXObjTypeId_t objectType, /* INPUT */
    MWXIteratorId_t *iterId /* OUTPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell containing the object to be retrieved.
objectType	Specifies the object type name (from mwx.h) of the object to be retrieved.
iterId	Specifies the iterator identifier.

Return

MWXSuccess	Occurs if the iterator was successfully returned.
MWXFail	Occurs if the iterator was not successfully returned.

Example

```
int queryNetName (
    MWXCellId_t      cellId
    MWXObjId_t       netId
)
{
    MWXIteratorId_t  netIterId;
    char             *netName;
    int              rc;

    if (MWXDb_Iter_ObjectByType( cellId, MWX_NET, &netIterId) != MWXSuccess) {
        MWX_Printf((char *)"Failed to iterate net object\n");
        return 0;
    }

    while ((rc = MWXDb_Iter_Next( netIterId, &netId)) == MWXSuccess) {
        if (MWXDb_Get_Net_name( cellId, netId, &netName) != MWXSuccess) {
            MWX_Printf((char*) "Fail to get net name\n");
        }
        MWX_Printf ((char *)"Net name = %s\n", netName);
    }

    if (rc == MWXIterDone)
        MWX_Printf((char *)"Done with net object iteration.\n");
    else
        MWX_Printf((char *)"Failed during the iteration of net object\n");

    MWXDb_Iter_Quit( netIterId);
    return 1;
}
```

See Also

[MWXDb_Iter_Next](#)
[MWXDb_Iter_Quit](#)

MWXDb_Iter_Quit

Terminates the object iteration by a specified iterator ID.

Syntax

```
int MWXDb_Iter_Quit (
    MWXIteratorId_t iterId, /* INPUT */
)
```

Parameters

iterId	Specifies the iterator identifier.

Return

MWXSuccess	Occurs if the iterator is successfully released.

MWXFail	Occurs if the iterator is not successfully released.

Example

```
int queryNetName (
    MWXCellId_t      cellId
    MWXObjId_t       netId
)
{
    MWXIteratorId_t  netIterId;
    char             *netName;
    int              rc;

    if (MWXDb_Iter_ObjectByType( cellId, MWX_NET, &netIterId) != MWXSuccess) {
        MWX_Printf((char *)"Failed to iterate net object\n");
        return 0;
    }

    while ((rc = MWXDb_Iter_Next( netIterId, &netId)) == MWXSuccess) {
        if (MWXDb_Get_Net_name( cellId, netId, &netName) != MWXSuccess) {
            MWX_Printf((char*) "Fail to get net name\n");
        }
        MWX_Printf ((char *)"Net name = %s\n", netName);
    }

    if (rc == MWXIterDone)
        MWX_Printf((char *)"Done with net object iteration.\n");
    else
        MWX_Printf((char *)"Failed during the iteration of net object\n");

    MWXDb_Iter_Quit( netIterId);
    return 1;
}
```

See Also

[MWXDb_Iter_ObjectByType](#)
[MWXDb_Iter_Next](#)

MWXDb_Iter_Quit_LibCellViewPair

Quit to iterates all cell-view name pairs in an opened library or plus all its reference libraries.

Syntax

```
int32 MWXDb_Iter_Quit_LibCellViewPair(  
    MWXIteratorId_t iterId /* INPUT */  
)
```

Parameters

iterId A returned iterator Id.

Return

MWXSuccess Occurs if successfully quit to iterating.

MWXFail Occurs if fail to quit iterating.

Example

See example in [MWXDb_Iter_Next_LibCellViewPair](#)

See Also

[MWXDb_Iter_LibCellViewPair](#)
[MWXDb_Iter_Next_LibCellViewPair](#)

MWXDb_Merge_Objects

Merges all objects on the same layer in the input object array (objectsArray) that are contained in the cell (cellId) to produce an array of merged objects (mergedObjects). This layer can also contain mergedObjCount objects on it. The input object's array must end with a 0. It is your responsibility to free the memory of the return object's array.

Syntax

```
int MWXDb_Merge_Objects (MWXCellId_t cellId,  
    /* INPUT : cell identification number. */  
    MWXObjId_t *objectsArray, /* INPUT : array of
```

MWXDb_Iter_Quit_LibCellViewPair

```

        objects. */
int32 *mergedObjCount, /* OUTPUT : number of
        objects in the merged array. */
MWXObjId_t **mergedObjects /* OUTPUT : array of
        merged objects. */
)

```

Parameters

cellId	Specifies the database identifier of the cell containing the objects to merge.
objectsArray	Specifies the given array of the objects to merge.
mergedObjCount	Specifies the number of objects in the resulting merged array of objects.
mergedObjects	Specifies the resulting merged array of objects.

Return

MWXSuccess	Occurs if all the objects in the given object's array are successfully merged.
MWXFail	Occurs if all the objects in the given object's array are not successfully merged.

Example

```

rc = MWXDb_Merge_Objects(cellId, objectsArray, mergedObjCount, &mergedObjects);
switch (rc) {
    case MWXSuccess:
        return (MWXSuccess);
    case MWXFail:
        MWX_Printf("Fail to merge all objects of the
        given objects' array. \n");
        return (MWXFail);
    default:
        return (MWXFail);
}

```

MWXDb_omwxInit

This function is only used in conjunction with stand-alone programs. It initializes the Milkyway database and checks out an appropriate run-time license. This function must be called before any other Milkyway API functions. Functions will return the code `MWXNoLicense` if `MWXDb_omwxInit` is not called first.

Syntax

```
int MWXDb_omwxInit (
    int argc,          /* INPUT */
    char **argv,        /* INPUT */
    int caseSensitive /* INPUT */
)
```

Parameters

<code>argc</code>	Specifies the number of command line arguments.
<code>**argv</code>	Contains the command line arguments list.
<code>caseSensitive</code>	Specifies case sensitivity, 0 if not case sensitive, otherwise case sensitive.

Return

<code>MWXSuccess</code>	Occurs if the initialization is successful.
	The main program exits if initialization is not successful.

Example

```
int rc; /* return code */
int caseSensitive = 0; /* NOT case sensitive */

rc = MWXDb_initOmwx ( argc, argv, caseSensitive );
/* Does not return if initialization failed. */
/* All other Milkyway API functions go after this */
```

MWXDb_Open_Cell

Opens a cell for read or write, depending on the access mode supplied. The database identifier of the cell is returned in `cellId`. The cell ID is undefined if the return code is not `MWXSuccess` or `MWXCellAlreadyOpen`.

Syntax

```
int MWXDb_Open_Cell(
    char *cellName, /* INPUT */
    char accessMode, /* INPUT */
    MWXCellId_t *cellId /* OUTPUT */
)
```

Parameters

<code>cellName</code>	Specifies the name of the cell to open.
<code>accessMode</code>	Specifies a <code>w</code> for write access or an <code>r</code> for read access.
<code>cellId</code>	Specifies the database identifier of the cell that was opened.

Return

<code>MWXSuccess</code>	Occurs if the cell is successfully opened.
<code>MWXNoLibraryOpened</code>	Occurs if no library is open.
<code>MWXCellAlreadyOpened</code>	Occurs if the cell is already open.
<code>MWXFail</code>	Occurs if the cell cannot be opened for any other reason.

Example

```
rc = MWXDb_Open_Cell (cellName, mode, &cellId);
    switch (rc) {
        case MWXSuccess:
            break;
        case MWXFail:
            MWX_Printf("ERROR -- Failed to open cell
                '%s'\n", cellName);
            break;
        case MWXCellAlreadyOpened:
            MWX_Printf("ERROR -- Cell '%s' is already
                open\n", cellName);
            break;
        case MWXNoLibraryOpened:
            MWX_Printf("ERROR -- No library is open\n");
            break;
    }
```

Scheme Equivalent

`dbOpenCell`

(see Physical Implementation Online Help)

MWXDb_Open_CellAttachFile

Opens an attached file of a cell in the open library.

Syntax

```
FILE *                               MWXDb_Open_CellAttachFile (
                                     MWXCellId_t cellId, /* INPUT */
                                     char *pseudoAccessName, /* INPUT */
                                     char *type /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell whose attached file is to be opened.
pseudoAccessName	Identifies the attached file to be opened.
type	The mode in which the attached file is to be opened. The modes available are the same as UNIX <code>fopen()</code> .

Return

FILE *	A file pointer is returned if the file is successfully opened.
NULL(0)	Occurs if the attached file cannot be successfully opened.

Example

```
fp = MWXDb_Open_CellAttachFile ( cellId, pseudoAccessName,
                                type);
if( fp != NULL )
{
    MWX_Printf("INFO -- File '%s' opened\n",fileName);
}
else
{
    MWX_Printf("ERROR -- Failed to open cell attached file\n");
    return (MWXFail);
}
```

MWXDb_Open_Cell_ByLibId

Opens a cell by a given library ID (`libId`) for read or write depending on the access mode supplied. The returned cell ID (`cellId`) is undefined if the return code is not `MWXSuccess` or `MWXCellAlreadyOpen`.

Syntax

```
int MWXDb_Open_Cell_ByLibId (
    MWXLibId_t libId, /* INPUT */
    char *cellName, /* INPUT */
    char accessMode, /* INPUT */
    MWXCellId_t *cellId /* OUTPUT */
)
```

)

Parameters

<code>libId</code>	Specifies the library containing the cell.
<code>cellName</code>	Specifies the cell to open.
<code>accessMode</code>	Specifies <code>w</code> for write access or <code>r</code> for read access.
<code>cellId</code>	Returned database ID number of the cell just opened.

Return

<code>MWXSucces</code>	Occurs if <code>libId</code> and <code>cellName</code> open the cell successfully.
<code>MWXCellAlreadyOpen</code>	Occurs if the cell is already open.
<code>MWXFail</code>	Occurs if <code>libId</code> does not open the cell successfully.

Dynamic Example

```
#include<stdio.h>
#include<stdlib.h>
#include "mwX.h"
#include "mwX_proto.h"

MWXSCM_t mwOpenCellByLibId (MWXSCM_t list)
{
    char progDescrString[50] = "Open Cell By LibId -";
    char libName[256];
    char cellName[256];
    char accessMode;
    MWXCellId_t cellId;
    MWXLibId_t libId;
    int count, i;
    static int caseSensitive = 0;
    /*-----
    GET THE SCHEME LIST
    -----*/
    if (MWXSc_Scanf (list, "ss",libName,cellName) != 2) {
        MWX_Printf((char*)"Error.Synopsis : openCellByLibId libName cellName\n");
        return (MWXFail);
    }
    /*-----
    HELLO
    -----*/
    MWX_Printf((char*)" \n%s Testing...\n", progDescrString);

    /* Get libId by library name */

    if (MWXDb_Get_LibId_ByLibName(libName, &libId) != MWXSucces) {
```

```

        MWX_Printf((char*) "Error : Can't get LibId by libName\n");
        return(MWXFail);
    }

    /* open cell by libId */

    if (MWXDb_Open_Cell_ByLibId(libId,cellName, 'w', &cellId) != MWXSuccess) {
        MWX_Printf((char*) "Error : Can't open cell %s using
MWXDb_Open_Cell_ByLibId\n",cellName);
        return (MWXFail);
    }

    /*-----
    DONE
    -----*/
    MWXDb_Close_Cell(cellId);
    MWXDb_Close_Lib();
    MWX_Printf((char*) "%s DONE TESTING! \n\n",progDescrString);
    return(MWXSuccess);
}
void reg_openCellByLibId(){
    MWXSc_Register_Function((char*) "openCellByLibId",mwOpenCellByLibId);
}

```

Stand-Alone Example

```

int OpenCellByLibId (char *libName, char *cellName)
{
    MWXCellId_t cellId;
    MWXLibId_t libId;

    /*-----
    HELLO
    -----*/

    /* Get libId by library name */

    if (MWXDb_Get_LibId_ByLibName(libName, &libId) != MWXSuccess) {
        MWX_Printf((char*) "Error : Can't get LibId by libName\n");
        return(MWXFail);
    }

    /* open cell by libId */

    if (MWXDb_Open_Cell_ByLibId(libId,cellName, 'w', &cellId) != MWXSuccess) {
        MWX_Printf((char*) "Error : Can't open cell %s using
MWXDb_Open_Cell_ByLibId\n",cellName);
        return (MWXFail);
    }

    MWXDb_Close_Cell(cellId);
    MWXDb_Close_Lib();
    return(MWXSuccess);
}

```

See Also

[MWXDb_Open_Cell](#)

MWXDb_Open_Lib

Opens a library for editing. If the library is successfully opened, the database identifier of the library will be returned in `libraryId` and its reference libraries will be opened in read mode.

Syntax

```
int MWXDb_Open_Lib(  
    char *libraryName, /* INPUT */  
    MWXLibId_t *libraryId /* OUTPUT */  
)
```

Parameters

libraryName	Specifies the name of the library to open.
libraryId	Returns the database identifier assigned to the opened library.

Return

MWXSucces	Occurs if the library is successfully opened.
MWXLlibraryAlreadyOpened	Occurs if the library is already open.
MWXFail	Occurs if the library cannot be successfully opened (for example, if it does not exist).

Dynamic Example

```
MWXSCH_t mwOpenLib(MWXSCH_t list) {  
  
    int rc;  
    char libraryName[256];  
    MWXLibId_t libraryId;  
  
    if (MWXSc_Scanf(list, "s", libraryName) != 1) {  
MWX_Printf("ERROR -- mwOpenLib requires 1 argument.\n");  
return (MWXFail);  
}  
  
MWX_Printf("INFO -- MWXDb_Open_Lib. Opening library %s.\n", libraryName);  
rc = MWXDb_Open_Lib(libraryName, &libraryId);  
  
    switch (rc) {  
        case MWXSucces:  
MWX_Printf("Successfully opened library. Library Id = %d.\n", libraryId);  
return (MWXSc_Make_DBIdSCH(libraryId));  
        case MWXLlibraryAlreadyOpened:  
MWX_Printf("ERROR -- Design library is already opened. MWXDb_Open_Lib  
failed.\n");  
return (MWXFail);  
        case MWXFail:  
MWX_Printf("ERROR -- Unable to open library. MWXDb_Open_Lib failed.\n");  
return (MWXFail);  
        default:  
MWX_Printf("UNKNOWN ERROR %d from MWXDb_Open_Lib.\n", rc);  
return (MWXFail);  
    }  
}
```

Stand-Alone Example

```
int OpenLib(char *libraryName, MWXLibId_t *libraryId) {  
  
    int rc;  
  
    MWX_Printf("INFO -- MWXDb_Open_Lib. Opening library %s.\n", libraryName);
```

```

rc = MWXDb_Open_Lib(libraryName, libraryId);

switch (rc) {
case MWXSuccess:
MWX_Printf("Successfully opened library.  Library Id = %d.\n", *libraryId);
return (MWXSuccess);
case MWXLibraryAlreadyOpened:
MWX_Printf("ERROR -- Design library is already opened.  MWXDb_Open_Lib
failed.\n");
return (MWXFail);
case MWXFail:
MWX_Printf("ERROR -- Unable to open library.  MWXDb_Open_Lib failed.\n");
return (MWXFail);
default:
MWX_Printf("UNKNOWN ERROR %d from MWXDb_Open_Lib.\n", rc);
return (MWXFail);
}
}

```

Scheme Equivalent

dbOpenLib

(see Physical Implementation Online Help)

See Also

[MWXDb_Create_Lib](#)

[MWXDb_Get_CurrentLibId](#)

[MWXDb_Rebuild_Library](#)

[MWXDb_Open_Library_ByAccessMode](#)

[MWXDb_Close_Lib](#)

MWXDb_Open_LibAttachFile

Opens an attached file of an open library.

Syntax

```

FILE *                               MWXDb_Open_LibAttachFile (
char *pseudoAccessName, /* INPUT */
char *type /* INPUT */
)

```

Parameters

<code>pseudoAccessName</code>	Identifies the attached file to be opened.
<code>type</code>	The mode in which the attached file is to be opened. The modes available are the same as UNIX <code>fopen()</code> .

Return

<code>FILE *</code>	A file is pointer returned if the file is successfully opened.
<code>0</code>	Occurs if the attached file cannot be successfully opened.

Example

```
fp = MWXDb_Open_LibAttachFile ( pseudoAccessName, type );
if( fp != NULL )
{
    MWX_Printf("INFO -- File '%s' opened\n",fileName);
}
else
{
    MWX_Printf("ERROR -- Failed to open lib attached file\n");
    return (MWXFail);
}
```

MWXDb_Open_Library_ByAccessMode

Similar to `MWXDb_Open_Lib` except the design library is opened in the given access mode.

Syntax

```
int MWXDb_Open_Library_ByAccessMode(
char *libName, /* INPUT */
char *accessMode, /* INPUT */
MWXLibId_t *libId /* OUTPUT */
)
```

Parameters

<code>libName</code>	Specifies the name of the library to open.
<code>accessMode</code>	Specifies the mode in which to open the library, "r" for read only or "w" for writable.
<code>libId</code>	Returns the database identifier assigned to the opened library.

Return

<code>MWXSucces</code>	Occurs if the library is opened in the given access mode.
<code>MWXLibraryAlreadyOpened</code>	Occurs if the design library is already open.
<code>MWXFail</code>	Occurs if the library fails to open for any reason.

Dynamic Example

```
MWXSCM_t mwOpenLibraryByAccessMode(MWXSCM_t list) {  
  
    int rc;  
    char libName[256];  
    char accessMode[10];  
    MWXLibId_t libId;  
  
    if (MWXSc_Scanf(list, "ss", libName, accessMode) != 2) {  
        MWX_Printf("ERROR -- mwOpenLibraryByAccessMode requires 2 arguments.\n");  
        return (MWXFail);  
    }  
  
    MWX_Printf("INFO -- MWXDb_Open_Library_ByAccessMode.\n");  
    rc = MWXDb_Open_Library_ByAccessMode(libName, accessMode, &libId);  
  
    switch (rc) {  
        case MWXSucces:  
            MWX_Printf("Successfully opened library in %c mode. Library Id = %d.\n",  
                      accessMode[0], libId);  
            return (MWXSc_Make_DBIIdSCM(libId));  
        case MWXLibraryAlreadyOpened:  
            MWX_Printf("ERROR -- Design library is already opened.  
MWXDb_Open_Library_ByAccessMode failed.\n");  
            return (MWXFail);  
        case MWXFail:  
            MWX_Printf("ERROR -- Unable to open library.  
MWXDb_Open_Library_ByAccessMode failed.\n");  
            return (MWXFail);  
        default:  
            MWX_Printf("UNKNOWN ERROR %d from MWXDb_Open_Library_ByAccessMode.\n",  
                      rc);  
            return (MWXFail);  
    }  
}
```



```
}
```

Stand-Alone Example

```
int OpenLibraryByAccessMode(char *libName, char *accessMode, MWXLibId_t *libId) {

int rc;

MWX_Printf("INFO -- MWXDb_Open_Library_ByAccessMode.\n");
rc = MWXDb_Open_Library_ByAccessMode(libName, accessMode, libId);

switch (rc) {
case MWXSuccess:
MWX_Printf("Successfully opened library in %c mode. Library Id = %d.\n",
          accessMode[0], *libId);
return (MWXSuccess);
case MWXLibraryAlreadyOpened:
MWX_Printf("ERROR -- A library is already opened.
MWXDb_Open_Library_ByAccessMode failed.\n");
return (MWXFail);
case MWXFail:
MWX_Printf("ERROR -- Unable to open library.
MWXDb_Open_Library_ByAccessMode
          failed.\n");
return (MWXFail);
default:
MWX_Printf("UNKNOWN ERROR %d from
          MWXDb_Open_Library_ByAccessMode.\n", rc);
return (MWXFail);
}
}
```

See Also

[MWXDb_Create_Lib](#)
[MWXDb_Get_CurrentLibId](#)
[MWXDb_Open_Lib](#)
[MWXDb_Close_Lib](#)

MWXDb_Open_RefLibAttachFile

Opens an attached file of a reference library.

Syntax

```
int MWXDb_Open_RefLibAttachFile (
    MWXLibId_t    libraryId,                /* INPUT */
    char          *pseudoAccessName, /* INPUT */
    char          *type/* INPUT */
)
```

Parameters

<code>libraryId</code>	Specifies the database identifier of the reference library whose attached file is to be opened.
<code>pseudoAccessName</code>	Identifies the attached file to be opened.
<code>type</code>	The mode in which the attached file is to be opened. The modes available are the same as UNIX <code>fopen()</code> .

Return

<code>FILE *</code>	A file pointer is returned if the file is successfully opened.
<code>0</code>	Occurs if attached file cannot be successfully opened.

Example

```
fp = MWXDb_Open_RefLibAttachFile ( libraryId, pseudoAccessName, type );
    if( fp != NULL )
    {
        MWX_Printf("INFO -- File '%s' Opend\n",pseudoAccessName);
    }
    else
    {
        MWX_Printf("ERROR -- Failed to Open reference lib attach file\n");}
```

MWXDb_Output_ClickedPoint

Outputs the coordinate data of the clicked point from the mouse clicked point.

Syntax

```
int MWXDb_Get_CaseSensitive(
MWXPoint_t *inputPoint /* OUTPUT */
)
```

Parameters

`inputPoint` The clicked point on the open cell screen

Return

`MWXSuccess` Occurs if successfully gets the clicked point

`MWXFail` Occurs if fail to get clicked point

Example

```
rc = MWXDb_Output_ClickedPoint(&inputPoint);
switch (rc) {
    case MWXSuccess:
        printf("The coordinate of the clicked point is (%d %d) \n",
inputPoint.x, inputPoint.y);
        break;
    case MWXFail:
        printf("Fail to get the coordinate of the clicked point\n");
        break;
    Default:
        printf("Unknown return code \n");
        break;
}
```

MWXDb_Place_CellInstanceInLegalLoc

Places the cell instance in a legal location inside the left, bottom, right, and top. Works only for cell instances (module/standard/macros).

Syntax

```
int32 MWXDb_Place_CellInstanceInLegalLoc(
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t cellInstanceId, /*INPUT */
    int32 left, * INPUT */
    int32 bottom, /* INPUT */
    int32 right, /* INPUT */
    int32 top, /* INPUT */
    uchar transform, /* INPUT */
    int32 *xLoc, /* INPUT/OUTPUT: */
    int32 *yLoc /* INPUT/OUTPUT: */
)
```

Parameters

cellId	Specifies the database identifier of the cell that was opened.
cellInstancedId	Specifies the database identifier of the cellInstance.
left	Specifies the x-coordinate for the left side of the bounding box.
bottom	Specifies the y-coordinate for the bottom of the bounding box.
right	Specifies the x-coordinate for the right side of the bounding box.
top	Specifies the y-coordinate for the top of the bounding box.
transform	Specifies the orientation of the cell instance (rotated, flipped, and so on).
xLoc	Specifies the initial x location; a successful return changes it to the final location.
yLoc	Specifies the initial y location; a successful return changes it to the final location.

Return

MWXSuccess	Occurs if the CellInstance is successfully placed in the legal location.
MWXFail	Occurs if the CellInstance is not successfully placed in the legal location.

Example

```
rc = MWXDb_Place_CellInstanceInLegalLoc (cellId, cellInstId, left, bottom,
right, top, transform, &xLoc, &yLoc);
    switch (rc) {
        case MWXSuccess:
            MWX_Printf("CellInstanceInLegalLoc      :  (%d,%d) \n", xLoc, yLoc);
            rtn = MWXSuccess;
            break;
        case MWXFail:
            rtn = MWXFail;
            break;
        default:
            rtn = MWXFail;
    }
    return (rtn);
}
```

MWXdB_Purge_AllPlanGroup

Deletes all the floorplan groups from the cell.

Syntax

```
int MWXdB_Purge_AllPlanGroup (
    MWXCellId_t cellId /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that owns the floorplan groups.

Return

MWXSuccess	Occurs if all the floorplan groups are successfully deleted from the cell.

MWXFail	Occurs if any of the floorplan groups cannot be successfully deleted from the cell.

Example

```
rc = MWXdB_Purge_AllPlanGroup( cellId )
switch (rc) {
    case MWXSuccess:
        return (MWXSuccess);
    case MWXFail:
        return (MWXFail);
}
```

Scheme Equivalent

dbPurgeAllPlanGroup

(see Physical Implementation Online Help)

MWXdB_Purge_AllSingleNet

Purges all single-port (or port instance) nets in the specified cell.

Syntax

```
int MWXdB_Purge_AllSingleNet (
    MWXCellId_t cellId /* INPUT */
)
```

Parameters

`cellId` Database object ID of the cell to which the information applies.

Return

`MWXFail` Occurs if all single-port nets are not successfully purged.

`MWXSuccess` Occurs if all single-port nets are successfully purged.

Example

```
rc = MWXDb_Purge_AllSingleNet(cellId);

switch (rc) {
    case MWXSuccess:
        return (MWXSuccess);
    case MWXFail:
        return (MWXFail);
    default:
        return (MWXFail);
}
```

Scheme Equivalent

`dbPurgeAllSingleNet`

(see Physical Implementation Online Help)

MWXDb_Purge_AllTdfPads

Deletes all pad constraints from the cell.

Syntax

```
int MWXDb_Purge_AllTdfPads (
    MWXCellId_t cellId /*INPUT*/
)
```

Parameters

`cellId` Specifies the database identifier of the cell that owns the pads.

Return

`MWXSuccess` Occurs if the pad constraint information for the pads is successfully deleted from the cell or no pad constraint is loaded to the pads of the cell.

`MWXFail` Occurs if the pad constraint information cannot be successfully deleted from the cell.

Example

```
rc = MWXDb_Purge_AllTdfPads ( cellId );
switch (rc) {
    case MWXSuccess:
        return (MWXSuccess)
    case MWXFail:
        MWX_Printf("ERROR -- Failed to retrieve the
                    wireMaster\n");
        return (MWXFail);
}
```

MWXDb_Purge_CellInstance_ByName

Purges a cell instance record from the database.

Syntax

```
int MWXDb_Purge_CellInstance_ByName (
    MWXCellId_t cellId, /* INPUT */
    char *instanceName /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell from which to purge the instance.
instanceName	Specifies the name of the cell instance to purge.

Return

MWXSuccess	Occurs if the cell instance is successfully purged.
MWXFail	Occurs if the cell instance is not successfully purged.

Example

```
rc = MWXDb_Purge_CellInstance_ByName (cellId, instanceName);
switch (rc) {
    case MWXSuccess:
        return (MWXSuccess);
    default:
        MWX_Printf("ERROR -- Failed to purge cell instance\n");
        return (MWXFail);
}
```

Scheme Equivalent

dbPurgeCellInstByName
(see Physical Implementation Online Help)

MWXDb_Purge_CellInstMaster

Deletes a cell instance master record from database. A side effect of this function is to purge all cell instances in this master.

Syntax

```
int MWXDb_Purge_CellInstMaster (
    MWXCellId_t cellId, /* INPUT */
    char *ciMasterName /* INPUT */
)
```


Parameters

<code>cellId</code>	Specifies the database identifier of the cell.
<code>ciMasterName</code>	Specifies the name of the cell instance master.

Return

<code>MWXFail</code>	Occurs if the cell instance master and its instances cannot be successfully purged.
<code>MWXSuccess</code>	Occurs if the cell instance master and its instances are successfully purged.

Example

```
rc = MWXDb_Purge_CellInstMaster ( cellId, ciMasterName );
switch (rc){
    case MWXSuccess:
        return (MWXSuccess)
    case MWXFail:
        MWX_Printf("ERROR -- Failed to operate. \n");
        return (MWXFail);
}
```

Scheme Equivalent

`dbPurgeCellInstMaster`
(see Physical Implementation Online Help)

MWXDb_Purge_Group_ByName

Purges a specified group from a cell. The cell is identified by `cellId`.

Syntax

```
int MWXDb_Purge_Group_ByName (
    MWXCellId_t cellId, /* INPUT */
    int32 groupSignature, /* INPUT */
    char *groupName /* INPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell that owns the group.
<code>groupSignature</code>	Specifies a group number. Valid values are 0–255. Synopsys reserves the numbers from 0–200 and 255.
<code>groupName</code>	Name of the group.

Return

<code>MWXNoLicense</code>	Occurs if the Synopsys license is not present.
<code>MWXSuccess</code>	Occurs if the group is successfully deleted.
<code>MWXFail</code>	Occurs if the group cannot be successfully deleted.

Example

```
rc = MWXDb_Purge_Group_ByName( cellId, groupSignature,
                               groupName );
switch (rc) {
    case MWXSuccess:
return (MWXSuccess);
    case MWXFail:
return (MWXFail);
}
```

MWXDb_Purge_Net

Removes the specified net from the database. The cell owning the net must be open for the net to be successfully removed.

Syntax

```
int MWXDb_Purge_Net (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t netId/* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that owns the net to delete.
netId	Specifies the database identifier of the net to remove.

Return

MWXSuccess	Occurs if the net is successfully removed.
MWXFail	Occurs if the net removal fails.

Example

```
rc = MWXDb_Purge_Net (cellId, netId);
if (rc == MWXSuccess) {
    return (MWXSuccess);
} else {
    return (MWXFail);
}
```

Scheme Equivalent

dbPurgeNet
(see Physical Implementation Online Help)

MWXDb_Purge_SingleObjectType

Purges a SingleObject by objectType.

Syntax

```
public int32
    MWXDb_Purge_SingleObjectType (
        MWXCellId_t cellId, /* INPUT */
        MWXObjTypeId_t objectType /*
            INPUT: only get this type of objects */
    )
```

Parameters

cellId	Specifies the database identifier of the cell containing the Instances in the specified BBox.
objectType	The following object types are supported: MWX_BASEARRAYMWX_CELLREGIONMWX_CELLROWMWX_TILEPA TTERNMWX_WIRETRACKMWX_WIREDIRECTION

Return

MWXSuccess	Occurs if objects were successfully purged.
MWXFail	Occurs if objects were not successfully purged.

Dynamic Example

```
MWXSCM_t mwPurgeSingleObjectType (MWXSCM_t list)
{
    MWXCellId_t cellId;
    MWXObjTypeId_t objectType;
    int32 rc, rtn;
    if (MWXSc_Scanf(list, "oo", &cellId, &objectType) != 2) {
        MWX_Printf("ERROR -- mwCreatePath requires 2 arguments\n");
        return (MWXFail);
    }
    switch (objectType) {
        case MWX_BASEARRAY      :
        case MWX_CELLREGION     :
        case MWX_CELLROW        :
        case MWX_TILEPATTERN    :
        case MWX_WIRETRACK      :
        case MWX_WIREDIRECTION:
            break;
        default :
            MWX_Printf("INFO : Invalid object Type\n");
            return MWXFail;
            break;
    }
    rc = MWXDb_Purge_SingleObjectType(cellId, objectType);
    switch (rc) {
        case MWXSuccess:
            rtn = MWXSuccess;
            break;
        case MWXFail:
            rtn = MWXFail;
            break;
        default:
            rtn = MWXFail;
    }
    return (rtn);
}
/* Define a function to register the new command. */
void scmTestInitFunc()
```

```

{
    MWXSc_Register_Function ("mwPurgeSingleObjectType",
mwPurgeSingleObjectType);
}

```

MWXDb_Rebuild_CellWithVersion

Rebuilds the cell by the given cellName, viewName, and version.

Note:

This function will work on the READ ONLY file if the new cell is keeping the same cellName, viewName, and version. If the new cell is changing to a new name, then it must have WRITE permission.

Syntax

```

public int
MWXDb_Rebuild_CellWithVersion(
    MWXLibId_t libId, /* INPUT */
    char *cellName, /* INPUT */
    char *viewName, /* INPUT */
    int32 version /* INPUT */
)

```

Parameters

libId	Specifies the design library database identification.
cellName	Specifies the name of the cell that is going to be rebuilt.
viewName	Specifies the name of the cell view that is going to be rebuilt.
version	Specifies the version number of the cell that is going to be rebuilt.

Return

MWXSuccess	Occurs when the cell is successfully rebuilt.
MWXFail	Occurs when Milkyway fails to rebuild the cell.

Example

```

void rebuild_cell (char *libName) {
    char viewName[256] = "CEL";
    char cellName[256] = "A";
    MWXLibId_t libId;
    MWXCellId_t cellId;
    int rc;
}

```

```

system("cp A:1 testlib/CEL/.");
system("chmod 444 testlib/CEL/A:1");

rc = MWXDb_Open_Lib(libName,&libId);
if (rc != MWXSuccess) {
    MWX_Printf("FAIL - Can't open library\n");
    exit(MWXFail);
}

    MWX_Printf("TESTING REBUILD CELL VERSION \n");
    MWX_Printf("Checking adding new cell with same name  READ
                ONLY MODE\n");
    rc = MWXDb_Rebuild_CellVersion(libId,cellName,
        viewName,1);
if (rc != MWXSuccess) {
    MWX_Printf("FAIL - Fail to call rebuild cell version
                function\n");
    exit(MWXFail);
}
system("chmod 755 testlib/CEL/A:1");
rc = MWXDb_Check_Cell_Exist(cellName);
if (rc != MWXSuccess) {
    MWX_Printf("FAIL -- New cell is not existed in the
                library - Rebuild failed\n");
    exit(MWXFail);
}
MWX_Printf("SUCCESSFUL REBUILD LIBRARY\n");
}

```

MWXDb_Rebuild_Library

Rebuilds the specified library so that only changes to the library that do not come from the internal database functions are reflected.

Syntax

```

int MWXDb_Rebuild_Library(
char *libName /* INPUT */
)

```

Parameters

`libName` Name of the library that you want to rebuild.

Return

`MWXSuccess` Occurs if the library rebuild is successful.

`MWXFail` Occurs if the library rebuild fails.

Description

The `MWXDb_Rebuild_Library` function rebuilds the specified library so that only changes to the library that do not come from the internal database functions are reflected. For example, you can call this function to refresh the list of valid cell names in the library after you copy or delete a cell into or from the library's directory using the UNIX `cp` or `rm` command. The library you are trying to rebuild must be closed.

Dynamic Example

```
MWXSCM_t mwRebuildLibrary(MWXSCM_t list) {  
  
    int rc;  
    char libName[256];  
  
    if (MWXSc_Scanf(list, "s", libName) != 1) {  
MWX_Printf("ERROR -- mwRebuildLibrary requires 1 argument.\n");  
return (MWXFail);  
    }  
  
    MWX_Printf("INFO -- MWXDb_Rebuild_Library.\n");  
    rc = MWXDb_Rebuild_Library(libName);  
  
    switch (rc) {  
case MWXSuccess:  
        MWX_Printf("Successfully rebuilt library.");  
        return (MWXSuccess);  
case MWXFail:  
MWX_Printf("ERROR -- Unable to rebuild library.  MWXDb_Rebuild_Library  
        failed.\n");  
        return (MWXFail);  
default:  
MWX_Printf("UNKNOWN ERROR %d from MWXDb_Rebuild_Library.\n", rc);  
return (MWXFail);  
    }  
}
```

Stand-Alone Example

```
int RebuildLibrary(char *libName) {  
  
    int rc;  
  
    MWX_Printf("INFO -- MWXDb_Rebuild_Library.\n");
```

```

rc = MWXDb_Rebuild_Library(libName);

switch (rc) {
case MWXSuccess:
    MWX_Printf("Successfully rebuilt library.");
    return (MWXSuccess);
case MWXFail:
    MWX_Printf("ERROR -- Unable to rebuild library.  MWXDb_Rebuild_Library
              failed.\n");
    return (MWXFail);
default:
    MWX_Printf("UNKNOWN ERROR %d from MWXDb_Rebuild_Library.\n", rc);
    return (MWXFail);
}
}

```

See Also

[MWXDb_Create_Lib](#)
[MWXDb_Get_CurrentLibId](#)
[MWXDb_Open_Lib](#)
[MWXDb_Open_Library_ByAccessMode](#)
[MWXDb_Close_Lib](#)

MWXDb_Rebuild_Library_WithAccessCheck

Rebuilds the library with the access check for the Read only file. This also allows you to create new views in the library.

Note:

This function will work on the READ ONLY file if the new cell checked will keep the same cellName, viewName, and version. If the new cell checked is changing to a new name, then it must have WRITE permission.

Syntax

```

public int
MWXDb_Rebuild_Library_WithAccessCheck(
    char    *libName    /* INPUT */
    char    *viewNames  /* INPUT */
)

```


Parameters

libName	Specifies the name of the library to be rebuilt.
viewNames	Specifies the string of view names separated with a space, for example: "DIODE FRAM CEL"

Return

MWXSuccess	Occurs when the library is successfully rebuilt.
MWXFail	Occurs when Milkyway fails to rebuild the library.

Example

```
int main(int argc, char **argv) {
    static int caseSensitive = 0;
    char viewName[256] = "CEL";
    char *cellName = "A";
    MWXLibId_t libId;
    MWXCellId_t cellId;
    int rc;

    system("cp testing/testlib/CEL/A:1 testlib/CEL/.");
    system("chmod 444 testlib/CEL/A:1");
    rc = MWXDb_Rebuild_Library_WithAccessCheck
        (libName,viewName);

    if (rc != MWXSuccess) {
        MWX_Printf("FAIL -- Fail to call function ");
        MWX_Printf("MWXDb_Rebuild_Library_WithAccessCheck \n");
        exit (MWXFail);
    }
    else {
        MWX_Printf("Checking the library after rebuild\n");
        rc = MWXDb_Open_Lib(argv[1],&libId);
        if (rc != MWXSuccess) {
            MWX_Printf("FAIL - Can't open library\n");
            exit(MWXFail);
        }
        rc = MWXDb_Open_Cell(cellName,'r",&cellId);
        if (rc != MWXSuccess) {
            MWX_Printf("FAIL -- Library doesn't reconize new
                        cell Name\n");
            exit(MWXFail);
        }
        else {
            MWX_Printf("SUCCESSFUL PASSED THE TEST OF REBUILD
                        LIBRARY\n");
            MWXDb_Close_Cell(cellId);
        }
        MWXDb_Close_Lib();
    }
}
```

MWXDb_Recompute_CellBBox

Recomputes the cell bounding box for a given cell.

Syntax

```
int32 MWXDb_Recompute_CellBBox (  
    MWXCellId_t cellId /* INPUT */  
)
```

Parameters

cellId	Specifies the ID of the cell to perform the bounding box recomputation.

Return

MWXSuccess	Occurs if the cell's bounding box is successfully recomputed.

MWXFail	Occurs if the cell's bounding box recomputation fails.

Dynamic Example

```
MWXSCM_t mwRecomputeCellBBox(MWXSCM_t list) {  
  
    MWXCellId_t cellId;  
    int rc;  
  
    if (MWXSc_Scanf(list, "o", &cellId) != 1) {  
        MWX_Printf("ERROR - Usage : mwRecomputeCellBBox cellId.\n");  
        return MWX_FALSE_SCM;  
    }  
  
    rc = MWXDb_Recompute_CellBBox(cellId);  
  
    switch (rc) {  
    case MWXSuccess:  
        MWX_Printf("Successfully recomputed cell bbox.\n");  
        return MWX_TRUE_SCM;  
    case MWXFail:  
        MWX_Printf("Failed to recompute cell bbox.\n");  
        return MWX_FALSE_SCM;  
    default:  
        MWX_Printf("ERROR -- Unknown return code %d. MWXDb_Recompute_CellBBox "  
            "failed.\n", rc);  
        return MWX_FALSE_SCM;  
    }  
}
```

Stand-Alone Example

```
int RecomputeCellBBox(MWXCellId_t cellId) {

    int rc;

    MWX_Printf("INFO -- MWXDb_Recompute_CellBBox.\n");
    rc = MWXDb_Recompute_CellBBox(cellId);

    switch (rc) {
    case MWXSuccess:
        MWX_Printf("Successfully recomputed cell bbox.\n");
        return MWXSuccess;
    case MWXFail:
        MWX_Printf("Failed to recompute cell bbox.\n");
        return MWXFail;
    default:
        MWX_Printf("ERROR -- Unknown return code %d. MWXDb_Recompute_CellBBox "
                   "failed.\n", rc);
        return MWXFail;
    }
}
```

MWXDb_Remove_Object_FromGroup

Removes an object from a group specified by `groupId`.

Syntax

```
int MWXDb_Remove_Object_FromGroup (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t objectId, /* INPUT */
    MWXObjId_t groupId /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that owns the object and the group.
objectId	Specifies the database identifier of the object that will be removed from the group.
groupId	Specifies the database identifier of the group from which the object is to be removed.

Return

MWXNoLicense	Occurs if the Synopsys license is not present.
MWXSuccess	Occurs if the object is successfully removed from the group.
MWXFail	Occurs if the object cannot successfully be removed from the group.

Example

```
rc = MWXDb_Remove_Object_FromGroup( cellId, groupId, objectId );
switch (rc) {
    case MWXSuccess:
return (MWXSuccess);
    case MWXFail:
return (MWXFail);
}
```

Scheme Equivalent

dbRemoveObjectFromGroup
(see Physical Implementation Online Help)

MWXDb_Rename_Cell

Renames the cell name.

Syntax

```
int MWXDb_Rename_Cell (
    char *cellName1, /* INPUT */
    char *cellName2 /* INPUT */
)
```

Parameters

cellName1 The original cell name.

cellName2 The new cell name.

Return

MWXNoLibraryOpened Occurs if the library is not open.

MWXSuccess Occurs if the cell is successfully renamed.

MWXFail Occurs if the cell cannot be successfully renamed.

Example

```
        rc = MWXDb_Rename_Cell ( cellName1, cellName2 );
        switch (rc) {
case MWXSuccess:
    return (MWXSuccess)
case MWXFail:
    MWX_Printf("ERROR -- Failed to operate. \n");
    return (MWXFail);
}
```

Scheme Equivalent

dbRenameCell

(see Physical Implementation Online Help)

MWXDb_Rename_Lib

Renames a library. The library being renamed must be closed before calling this function; otherwise, MWXFail is returned.

Syntax

```
int MWXDb_Rename_Lib (
    char *oldLibraryName, /* INPUT */
    char *newLibraryName /* INPUT */
)
```

Parameters

oldLibraryName	The original library name.
newLibraryName	The new library name.

Return

MWXSuccess	Occurs if the library is successfully renamed.
MWXFail	Occurs if the library is not successfully renamed.

Dynamic Example

```
MWXSCM_t mwRenameLib(MWXSCM_t list) {  
  
    int rc;  
    char oldLibraryName[256];  
    char newLibraryName[256];  
  
    if (MWXSc_Scanf(list, "ss", oldLibraryName, newLibraryName) != 2) {  
        MWX_Printf("ERROR -- mwRenameLib requires 2 arguments.\n");  
        return (MWXFail);  
    }  
  
    MWX_Printf("INFO -- MWXDb_Rename_Lib.\n");  
    rc = MWXDb_Rename_Lib(oldLibraryName, newLibraryName);  
  
    switch (rc) {  
        case MWXSuccess:  
            MWX_Printf("Successfully renamed library from %s to %s.\n", oldLibraryName,  
                      newLibraryName);  
            return (MWXSuccess);  
        case MWXFail:  
            MWX_Printf("ERROR -- Function MWXDb_Rename_Lib failed.\n");  
            return (MWXFail);  
        default:  
            MWX_Printf("UNKNOWN ERROR %d from MWXDb_Rename_Lib.\n", rc);  
            return (MWXFail);  
    }  
}
```

Stand-Alone Example

```
int RenameLib(char* oldLibraryName, char* newLibraryName) {  
    int rc;  
  
    MWX_Printf("INFO -- MWXDb_Rename_Lib.\n");  
    rc = MWXDb_Rename_Lib(oldLibraryName, newLibraryName);  
  
    switch (rc) {  
        case MWXSuccess:  
            MWX_Printf("Successfully renamed library from %s to %s.\n", oldLibraryName,
```

```

        newLibraryName);
return (MWXSuccess);
case MWXFail:
MWX_Printf("ERROR -- Function MWXDb_Rename_Lib failed.\n");
return (MWXFail);
default:
MWX_Printf("UNKNOWN ERROR %d from MWXDb_Rename_Lib.\n", rc);
return (MWXFail);
}
}

```

Scheme Equivalent

dbRenameLib

(see Physical Implementation Online Help)

See Also

[MWXDb_Rename_Cell](#)

MWXDb_Replace_CellInst

Substitutes a different cell for a particular cell instance. The new cell must be from the same library as the old cell and maintain the same location and orientation. Connections to the instance are maintained for those ports in the old cell with a port of the same name in the new cell.

Syntax

```

int MWXDb_Replace_CellInst (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t cellInstId, /* INPUT */
    char *newCellName, /* INPUT */
    char *newViewName /* INPUT */
)

```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell containing the instance to replace.
<code>cellInstId</code>	Specifies the database identifier of the cell instance to replace.
<code>newCellName</code>	Specifies the name of the cell to substitute for the cell instance identified by <code>cellInstId</code> .
<code>newViewName</code>	Specifies the name of the view of the cell to substitute for the cell instance identified by <code>cellInstId</code> .

Return

<code>MWXSuccess</code>	Occurs if the replacement is successful.
<code>MWXFail</code>	Occurs if the replacement fails.

Example

```
rc = MWXDb_Replace_CellInst (cellId, cellInstId, newCellName, newViewName);
if (rc != MWXSuccess) {
    MWX_Printf("ERROR -- Failed to replace cell instance\n");
}
```

Scheme Equivalent

`dbReplaceCellInst`
(see Physical Implementation Online Help)

MWXDb_Replace_CellTextChar

Changes all the characters appearing in `oldCharSet` to the single character appearing in `newChar` in all the text objects attached to a cell.

Syntax

```
int MWXDb_Replace_CellTextChar (
    MWXCellId_t cellId, /* INPUT */
    char *oldCharSet, /* INPUT */
    char *newChar /* INPUT */
)
```


Parameters

<code>cellId</code>	Specifies the database identifier of the cell containing text objects to change.
<code>oldCharSet</code>	Specifies a set of characters to change.
<code>newChar</code>	Specifies the character to which all characters in <code>oldCharSet</code> will be changed.

Return

<code>MWXSuccess</code>	Occurs if the replacement is successful.
<code>MWXFail</code>	Occurs if the replacement fails.

Example

```
rc = MWXDb_Replace_CellTextChar (cellId, oldCharSet, newChar);
if (rc != MWXSuccess) {
    return (MWXFail);
} else {
    return (MWXSuccess);
}
```

Scheme Equivalent

`dbReplaceCellTextChar`
(see Physical Implementation Online Help)

MWXDb_Replace_CellTextString

Changes the value of all text objects with the value appearing in `oldString` to the value appearing in `newString` in a cell.

Syntax

```
int MWXDb_Replace_CellTextString (
    MWXCellId_t cellId, /* INPUT */
    char *oldString, /* INPUT */
    char *newString /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell containing the text objects to change.
oldString	Specifies the value of the text objects to change.
newString	Specifies the new value to give to the text objects being changed.

Return

MWXSuccess	Occurs if the replacement is successful.
MWXFail	Occurs if the replacement fails.

Example

```
rc = MWXDb_Replace_CellTextString (cellId, oldString, newString);
if (rc != MWXSuccess) {
    return (MWXFail);
} else {
    return (MWXSuccess);
}
```

MWXDb_Replace_ChildCell

Substitutes a different cell for all instances of a particular cell. The new cell must be from the same library as the old cell and maintain the same locations and orientations of the old instances. Connections to the instances are maintained for those ports in the old cell with a port of the same name in the new cell.

Syntax

```
int MWXDb_Replace_ChildCell (
    MWXCellId_t cellId, /* INPUT */
    char *oldCellName, /* INPUT */
    char *oldViewName, /* INPUT */
    char *newCellName, /* INPUT */
    char *newViewName /* INPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell containing the instance to replace.
<code>oldCellName</code>	Specifies the name of the cell to replace.
<code>oldViewName</code>	Specifies the view name of the cell to replace.
<code>newCellName</code>	Specifies the name of the cell to substitute for the cell identified by <code>oldCellName</code> and <code>oldViewName</code> .
<code>newViewName</code>	Specifies the view name of the cell to substitute for the cell identified by <code>oldCellName</code> and <code>oldViewName</code> .

Return

<code>MWXSuccess</code>	Occurs if the replacement is successful.
<code>MWXFail</code>	Occurs if the replacement fails.

Example

```
rc = MWXDb_Replace_ChildCell (cellId, oldCellName, oldViewName, newCellName,
newViewName);
if (rc != MWXSuccess) {
    MWX_Printf("ERROR -- Failed to replace cell instances\n");
}
```

Scheme Equivalent

`dbReplaceChildCell`

(see Physical Implementation Online Help)

MWXDb_Replace_RefLib_ByName

Replaces an existing reference library in the open library.

Syntax

```
int MWXDb_Replace_RefLib_ByName (
    char *oldRefLibraryName, /* INPUT */
    char *newRefLibraryName /* INPUT */
)
```

Parameters

<code>oldRefLibraryName</code>	Name of the existing reference library to be replaced.
<code>newRefLibraryName</code>	Name of the new reference library.

Return

<code>MWXNoLibraryOpened</code>	Occurs if no library is open.
<code>MWXFail</code>	Occurs if the existing reference library is not successfully replaced.
<code>MWXSuccess</code>	Occurs if the existing reference library is successfully replaced.

Example

```
rc = MWXDb_Replace_RefLib_ByName(oldRefLibName, newRefLibName);
switch (rc) {
case MWXSuccess:
    return (MWXSuccess);
case MWXFail:
    return (MWXFail);
case MWXNoLibraryOpened:
    return (MWXSc_Make_IntegerSCM(rc));
default:
    return (MWXFail);
}
```

Scheme Equivalent

`dbReplaceRefLibByName`
(see Physical Implementation Online Help)

MWXDb_Replace_TechInfo

Replaces the existing technology file for an open library. You *must* close and reopen the library.

Syntax

```
int MWXDb_Replace_TechInfo (
    char *techFileName /* INPUT */
)
```

Parameters

techFileName	Name of the technology file that replaces the existing technology file's current library.

Return

MWXNoLibraryOpened	Occurs if no library is open.
MWXFail	Occurs if the existing technology file is not successfully replaced.
MWXSuccess	Occurs if the existing technology file is successfully replaced.

Example

```
rc = MWXDb_Replace_TechInfo(fileName);
switch (rc) {
    case MWXSuccess:
        return (MWXSuccess);
    case MWXFail:
        return (MWXFail);
    case MWXNoLibraryOpened:
        return (MWXSc_Make_IntegerSCM(rc));
    default:
        return (MWXFail);
}
```

Scheme Equivalent

dbReplaceTechFile
(see Physical Implementation Online Help)

MWXDb_Replace_TextChar

Changes all the characters appearing in `oldCharSet` to the single character appearing in `newChar` in all the text objects of all the cells in a library. All cells must be closed.

Syntax

```
int MWXDb_Replace_TextChar (
    MWXLibId_t libId, /* INPUT */
    char *oldCharSet, /* INPUT */
    char *newChar /* INPUT */
)
```

Parameters

<code>libId</code>	Specifies the database identifier of the library containing the cells to change.
<code>oldCharSet</code>	Specifies the set of characters to change.
<code>newChar</code>	Specifies the character to which all the characters in <code>oldCharSet</code> will be changed.

Return

<code>MWXSucces</code>	Occurs if the replacement is successful.
<code>MWXFail</code>	Occurs if the replacement fails.

Example

```
rc = MWXDb_Replace_TextChar (libId, oldCharSet, newChar);
if (rc != MWXSucces) {
    return (MWXFail);
} else {
    return (MWXSucces);
}
```

MWXDb_Replace_TextString

Changes the value of all text objects with the value appearing in `oldString` to the value appearing in `newString` in all cells appearing in a library.

Syntax

```
int MWXDb_Replace_TextString (
    MWXLibId_t libId, /* INPUT */
    char *oldString, /* INPUT */
    char *newString /* INPUT */
)
```

Parameters

libId	Specifies the database identifier of the library containing the cells to change.
oldString	Specifies the value of the text objects to change.
newString	Specifies the new value to give to the text objects being changed.

Return

MWXSucces	Occurs if the replacement is successful.
MWXFail	Occurs if the replacement fails.

Example

```
rc = MWXDb_Replace_TextString (libId, oldString, newString);
if (rc != MWXSucces) {
    return (MWXFail);
} else {
    return (MWXSucces);
}
```

MWXDb_Restore_Pin

Creates a pin or changes the existing pin with the given parameters.

Syntax

```
int MWXDb_Restore_Pin (
    MWXCellId_t cellId, /* INPUT */
    char *netName, /* INPUT */
    char *pinName, /* INPUT */
    int32 statusFlag, /* INPUT */
    int32 accessDir, /* INPUT */
    int32 statusFlag1, /* INPUT */
    int32 eeq, /* INPUT */
    int32 layerNumber, /* INPUT */
    int32 numberOfPoint, /* INPUT */
    MWXPoint_t pointArray[], /* INPUT */
    MWXObjId_t *pinId /* OUTPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell in which to restore pin.
<code>netName</code>	Specifies the name of net in the open cell.
<code>pinName</code>	Specifies the name of the pin to restore.
<code>statusFlag</code>	Specifies the status for placement. Valid values: 0 - unplaced, 1 - placed, and 2 - fixed.
<code>accessDir</code>	Specifies the direction of pin for routing.
<code>statusFlag1</code>	Specifies the status for soft-macro pins.
<code>eeq</code>	Specifies the EEQ class number.
<code>layerNumber</code>	Specifies the layer number of the pin.
<code>numberOfPoint</code>	Specifies the number of x-y coordinates pairs contained in the <code>pointArray</code> .
<code>pointArray</code>	Specifies the set of x-y coordinates pairs defining the pin boundary.
<code>pinId</code>	Specifies the database identifier of the pin.

Return

<code>MWXSuccess</code>	Occurs if pin is successfully restored.
<code>MWXFail</code>	Occurs if function fails to restore the pin.
<code>MWXNoLibraryOpened</code>	Occurs if there is no library currently open.

Example

```
rc = MWXDb_Restore_Pin( cellId, netName, pinName, statusFlag, accessDir,
    statusFlag1, eeq, layerNumber, numberOfPoints, pointArray, &objectId);

switch (rc) {
case MWXSuccess:
    return (MWXSc_Make_DBIIdSCM (objectId));
    return (MWXSuccess);
case MWXFail:
    MWX_Printf("ERROR -- Failed to restore pin\n");
    return (MWXFail);
default:
    MWX_Printf("ERROR -- unknown return code from MWXDb_Restore_Pin\n");
}
```



```

    return (MWXFail);
}

```

SCHEME EQUIVALENT

dbRestorePin

MWXDb_RollBack_Cell

Replaces the current cell with the last version of the cell saved to the disk.

Syntax

```

int MWXDb_RollBack_Cell (
MWXCellId_t cellId /* INPUT */
)

```

Parameters

cellId	Database object ID of the cell to be rolled back.

Return

MWXNoLibraryOpened	Occurs if the library is not open.
MWXFail	Occurs if the cell is not successfully replaced.
MWXSuccess	Occurs if the cell is successfully replaced.

Example

```

rc = MWXDb_RollBack_Cell(cellId);
switch (rc) {
    case MWXSuccess:
        return (MWXSuccess);
    case MWXFail:
        return (MWXFail);
    default:
        return (MWXFail);
}

```

Scheme Equivalent

dbRollBackCell (see *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Save_Cell

Saves the cell identified by cellId to disk.

Syntax

```
int MWXDb_Save_Cell(  
    MWCCellId_t cellId /* INPUT */  
)
```

Parameters

cellId Specifies the name of the cell to save.

Return

MWXSuccess Occurs if the cell is successfully saved.

MWXFail Occurs if the cell is not successfully saved.

Example

```
rc = MWXDb_Save_Cell (cellId);  
switch (rc) {  
    case MWXSuccess:  
        break;  
    default:  
        MWX_Printf("ERROR -- Failed to save cell\n");  
        break;  
}
```

Scheme Equivalent

dbSaveCell

(see Physical Implementation Online Help)

MWXDb_Set_CaseSensitive

This function sets the case sensitive mode.

Syntax

```
int MWXDb_Get_CaseSensitive(  
int mode        /* INPUT */  
)
```

Parameters

mode Case sensitive mode

Return

MWXSuccess Occurs if Milkyway successfully sets the case- sensitive mode.

MWXFail Occurs if Milkyway fails to set the case-sensitive mode

Example

```
rc = MWXDb_Set_CaseSensitive(mode);
switch (rc) {
    case MWXSuccess:
        printf("Successfully set the case sensitive mode");
        return;
    case MWXFail:
        printf("Fail to set the case sensitive mode\n");
        return 0;
    Default:
        printf("Unknown return code\n");
        return 0;
}
```

MWXDb_Set_CellInstanceFixStatus_ByName

Sets the fix status bits in the cell instance `statusFlag` field. See `aprCmdFixCell` in the *IC Layout Command Reference Manual* for further information regarding the `fixStatus` values.

Syntax

```
int MWXDb_Set_CellInstanceFixStatus_ByName(
    MWXCellId_t cellId, /* INPUT */
    char *cellInstanceName, /* INPUT */
    char *fixStatus /* INPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell containing the cell instance being processed.
<code>cellInstanceName</code>	Specifies the instance name of the cell whose status is being set.
<code>fixStatus</code>	Specifies the new status of the cell. Valid values are Unplaced, Placed, Fixed, and RigidFixed.

Return

<code>MWXSuccess</code>	Occurs if the status is successfully set.
<code>MWXFail</code>	Occurs if the status is not successfully set.

Example

```
rc = MWXDb_Set_CellInstanceFixStatus_ByName (cellId,
    cellInstanceName, &fixStatus);
switch (rc) {
    case MWXSuccess:
        break;
    case MWXFail:
        MWX_Printf("ERROR -- Failed to set Fix status\n");
        break;}
```

Scheme Equivalent

`dbSetCellInstFixStatusByName`
(see Physical Implementation Online Help)

MWXDb_Set_CellInstancePlacement_ByName

Updates the placement information for the cell instance with the name `cellInstanceName` in the cell identified by `cellId`.

Syntax

```
int MWXDb_Set_CellInstancePlacement_ByName(
    MWXCellId_t cellId, /* INPUT */
    char *cellInstanceName, /* INPUT */
    char *rotationStr, /* INPUT */
    char *mirrorStr, /* INPUT */
    char *justStr, /* INPUT */
    MWXPoint_t *placement /* INPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell containing the instance to place.
<code>cellInstanceName</code>	Specifies the name of the instance to which the placement changes apply.
<code>rotationStr</code>	Specifies the counterclockwise rotation value, in degrees (0, 90, 180, or 270).
<code>mirrorStr</code>	Specifies the direction, if any, in which to flip the cell instance (no, X, or Y).
<code>justStr</code>	Defines the placement of the cell instance around the origin. The valid values are as follows: <code>ll</code> - places the cell instance at the lower left of the origin <code>lc</code> - places the cell instance at the lower center of the origin <code>lr</code> - places the cell instance at the lower right of the origin <code>cl</code> - places the cell instance at the center left of the origin <code>cc</code> - centers the cell instance around the origin <code>cr</code> - places the cell instance at the center right of the origin <code>ul</code> - places the cell instance at the upper left of the origin <code>uc</code> - places the cell instance at the upper center of the origin <code>ur</code> - places the cell instance at the upper right of the origin
<code>placement</code>	Specifies the x/y-coordinate pair representing the location of the cell instance origin.

Return

<code>MWXSuccess</code>	Occurs if the cell instance is successfully updated.
<code>MWXFail</code>	Occurs if the cell instance is not successfully updated.

Example

```
/* Convert to database units. */
origin.x = (int32) x;
origin.y = (int32) y;
rc = MWXDb_Set_CellInstancePlacement_ByName (cellId, "myInstance",
    "90", "X", &origin);
switch (rc) {
    case MWXSuccess:
        return (MWXSuccess);
    default:
        MWX_Printf("ERROR -- Failed to update cell instance\n");
        return (MWXFail);
}
```

Scheme Equivalent

dbSetCellInstPlacement

(see Physical Implementation Online Help)

MWXdB_Set_CellLayoutSubType

Specifies a cell's layout subtype.

Syntax

```
int MWXdB_Set_CellLayoutSubType (
    MWXCellId_t cellId, /* INPUT */
    char *layoutSubType /* INPUT */
)
```

Parameters

cellId	Database identifier of the cell to which the information applies.
layoutSubType	Layout subtype that applies to the specified cell. Valid values are ioPad padFillerCell cornerPad standardCell standardCell2 stdFillerCell macro chip image cover unknown

Return

MWXNoLibraryOpened	Occurs if the library is not open.
MWXFail	Occurs if the layout subtype is not successfully specified.
MWXSuccess	Occurs if the layout subtype is successfully specified.

Example

```
switch (rc) {
case MWXSuccess:
    return (MWXSuccess);
case MWXFail:
    MWX_Printf("ERROR -- Failed to
                SetCellLayoutSubType\n");
    return (MWXFail);
default:
    MWX_Printf("ERROR -- unknown return
                code from SetCellLayoutSubType\n");
    return (MWXFail);
}
```

MWXdB_Set_CellLModelSubType

Specifies a cell's logic model subtype.

Syntax

```
int MWXdB_Set_CellLModelSubType (  
    MWXCellId_t cellId, /* INPUT */  
    char *lModelSubType /* INPUT */  
)
```

Parameters

cellId	Database identifier of the timing (.TIM) cell for which you are specifying a type.
lModelSubType	Cell type. Valid values include any of the following: latchflipflopromclock

Return

MWXNoLibraryOpened	Occurs if no library is open.
MWXFail	Occurs if the logic model subtype is not successfully specified.
MWXSuccess	Occurs if the logic model subtype is successfully specified.

Example

```
rc = MWXdB_Set_CellLModelSubType  
    (cellId,lModelSubType);  
MWX_Printf("INFO -- return value rc=%d\n",rc);  
switch (rc) {  
    case MWXSuccess:  
        return (MWXSuccess);  
    case MWXFail:  
        MWX_Printf("ERROR -- Failed to SetCellLModelSubType\n");  
        return (MWXFail);  
    default:  
        MWX_Printf("ERROR -- unknown return  
            code from SetCellLModelSubType\n");  
        return (MWXFail);  
}
```

MWXdB_Set_CellType

Sets or specifies the type of a cell.

Syntax

```
int MWXDb_Set_CellType (
    MWXCellId_t cellId, /* INPUT */
    char *cellType /* INPUT */
)
```

Parameters

cellId	Database object ID of the cell to which the information applies.
cellType	Type that applies to the cell. Valid values are maskLayoutlayoutnetListschematicssymbolicpcbLayo utlogicModelunknown

Return

MWXSuccess	Occurs if the cell types are successfully set.
MWXFail	Occurs if the cell types are not successfully set.

Example

```
    if (MWXDb_Set_CellType(cellId,cellType) != MWXSuccess) {
        MWX_Printf("Fail to set cell type\n");
        return (MWXFail);
    }
return (MWXSuccess);
```

Scheme Equivalent

dbSetCellType
(see Physical Implementation Online Help)

MWXDb_Set_CellType_Bit

Sets or specifies the CellType bit.

Syntax

```
int MWXDb_Set_CellType_Bit(
    MWXCellId_t cellId, /* INPUT */
    int32 bitConst /* INPUT */
)
```


Parameters

cellId	Database object ID of the cell to which the information applies.
bitConst	Bit constant that applies to the cell. Valid values are MWXcMaskLayoutViewTypeMWXcNetListViewTypeMWXcSchematicViewTypeMWXcSymbolicViewTypeMWXcLogicalModelViewType

Return

MWXSuccess	Occurs if the CellType bits are successfully set.
MWXFail	Occurs if the CellType bits are not successfully set.

Example

```
    if (MWXDb_Set_CellType_Bit(cellId,bitConst)!= MWXSuccess) {
MWX_Printf("Fail to set cell type bit\n");
    return (MWXFail);
}
return (MWXSuccess);
```

MWXDb_Set_ErrorObjStatus

Sets the value of the status field of an error object. The error status is not currently used by Synopsys tools.

Syntax

```
int MWXDb_Set_ErrorObjStatus (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t errorObjId, /* INPUT */
    uint32 status /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell owning the error object identified by errorObjId.
errorObjId	Specifies the database identifier of the error object being updated.
status	Specifies the value to assign to the status field of the error object.

Return

MWXSuccess	Occurs if Milkyway successfully sets the new status value.
MWXFail	Occurs if Milkyway fails to successfully set the new status value.

Example

```
    if (MWXDb_Create_ErrorType(cellId, "New Type", "Type message", 1,
        &errorTypeId) == MWXSuccess) {
        if (MWXDb_Create_ErrorObj(cellId, errorTypeId, "object message",
            &errorObjId) == MWXSuccess) {
            if (MWXDb_Set_ErrorObjStatus(cellId, errorObjId, 7) !=
MWXSuccess) {
                MWX_Printf("ERROR -- Failed to set error object
status\n");
            }
        } else {
            MWX_Printf("ERROR -- Failed to create error object\n"); }
    } else {
        MWX_Printf("ERROR -- Failed to create error type\n");
    }
}
```

Scheme Equivalent

dbSetErrorObjStatus

(see Physical Implementation Online Help)

MWXDb_Set_ErrorTypeStatus

Sets the value of the status field of an error type. The error status is not currently used by Synopsys tools.

Syntax

```
int MWXDb_Set_ErrorTypeStatus (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t errorTypeId, /* INPUT */
    uint16 status /* INPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell owning the error object identified by <code>errorTypeId</code> .
<code>errorTypeId</code>	Specifies the database identifier of the error type being updated.
<code>status</code>	Specifies the value to assign to the status field of the error type.

Return

<code>MWXSuccess</code>	Occurs if Milkyway successfully sets the new status value.
<code>MWXFail</code>	Occurs if Milkyway fails to successfully set the new status value.

Example

```
    if (MWXDb_Create_ErrorType(cellId, "New Type", "Type message", 1,
        &errorTypeId) == MWXSuccess) {
        if (MWXDb_Set_ErrorTypeStatus(cellId, errorTypeId, 7) !=
MWXSuccess) {
            MWX_Printf("ERROR -- Failed to set error type status\n");
        }
    } else {
        MWX_Printf("ERROR -- Failed to create error type\n");
    }
}
```

Scheme Equivalent

`dbSetErrorTypeStatus`

(see Physical Implementation Online Help)

MWXDb_Set_GetBBoxCallBack

Registers a callback function for the get input bbox from the mouse click.

Syntax

```
int MWXDb_Set_GetBBoxCallBack (
    int (*proc) (int *, int *, int *, int *)
)
```

Parameters

proc	Callback function.

Return

MWXSucccess	Occurs if the callback function is successfully registered.

MWXFail	Occurs if the call back function is not successfully registered.

Example

```
static int left;
static int bottom;
static int right;
static int top;

static int validate = 0;

int CheckBBox (
    int* l, int* b, int* r, int* t)
{
    if (( l!= NULL) && (b != NULL) && (r != NULL) && (t != NULL)) {
        validate = 1;
        lefl = *l;
        bottom = *b;
        right = *r;
        top = *t;
    }
    else validate = 0;
    return 1;
}

/*****
=
= Test Get Input BBox from mouse click
=
*****/

MWXSCM_t myGetInputBBox (MWXSCM_t list)
{
    char          shape[256];
    int32         rc;
    MWXSCM_t      rtn[2];

    if (MWXSc_Scanf(list, "s",shape) != 1) {
        MWX_Printf("Syntax : myGetInputBBox \"shape\" \n");
        return (MWXFail);
    }

    MWXDb_Set_GetBBoxCallBack (CheckBBox);
    if( MWXDb_Get_InputData(shape) != MWXSucccess) {
        MWX_Printf(" INFO : Can't get input bbox \n");
        return (MWXFail);
    }
}
```

```

}

/*****
=
= Test Output BBox from mouse click
=
=*****/
MWXSCM_t myOutputBBox (MWXSCM_t list)
{
    int32      rc;
    MWXSCM_t   rtn[2];

    MWX_Printf(" ***** Output point's coordinate ***** \n");
    if (validate) {
        rtn[0] = MWXSc_Make_IntegerSCM(left);
        rtn[1] = MWXSc_Make_IntegerSCM(bottom);
        rtn[2] = MWXSc_Make_IntegerSCM(right);
        rtn[3] = MWXSc_Make_IntegerSCM(top);
        validate = 0;
    }
    else {
        MWX_Printf(" INFO : Please get bbox first \n");
        return (MWXFail);
    }
    return (MWXSc_Make_ListSCM(4,rtn));
}

void scmTestInitFunc()
{
    MWX_Debug();
    MWXSc_Register_Function ("myGetInputBBox", myGetInputBBox);
    MWXSc_Register_Function ("myOutputBBox", myOutputBBox);
    MWX_Printf ("scmTestInitDone\n");
}

```

MWXDb_Set_GetLineCallBack

Registers a callback function for the get input line from the mouse click.

Syntax

```

int MWXDb_Set_GetLineCallBack (
    int (*proc) (int *, int *, int *, int *)
)

```

Parameters

`proc` Callback function.

Return

`MWXSuccess` Occurs if the callback function is successfully registered.

`MWXFail` Occurs if the call back function is not successfully registered.

Example

```
static int x11;
static int y11;
static int x22;
static int y22;

static int validate = 0;

int CheckLine (
    int* x1, int* y1, int* x2, int* y2)
{
    if ((x1 != NULL) && (y1 != NULL) && (x2 != NULL) && (y2 != NULL)) {
        validate = 1;
        x11 = *x1;
        y11 = *y1;
        x22 = *x2;
        y22 = *y2;
    }
    else validate = 0;
    return 1;
}

/*****
=
= Test Get Input Line from mouse click
=
*****/

MWXSCM_t myGetInputLine (MWXSCM_t list)
{
    char          shape[256];
    int32         rc;
    MWXSCM_t      rtn[2];

    if (MWXSc_Scanf(list, "s",shape) != 1) {
        MWX_Printf("Syntax : myGetInputLine \"shape\" \n");
        return (MWXFail);
    }

    MWXDb_Set_GetLineCallBack (CheckLine);
    if( MWXDb_Get_InputData(shape) != MWXSuccess) {
        MWX_Printf(" INFO : Can't get input line \n");
        return (MWXFail);
    }
}
```

```

}

/*****
=
= Test Output Line from mouse click
=
=*****/

MWXSCM_t myOutputLine (MWXSCM_t list)
{
    int32      rc;
    MWXSCM_t   rtn[4];

    MWX_Printf(" ***** Output Line's coordinate ***** \n");
    if (validate) {
        rtn[0] = MWXSc_Make_IntegerSCM(x11);
        rtn[1] = MWXSc_Make_IntegerSCM(y11);
        rtn[2] = MWXSc_Make_IntegerSCM(x22);
        rtn[3] = MWXSc_Make_IntegerSCM(y22);
        validate = 0;
    }
    else {
        MWX_Printf(" INFO : Please get line first \n");
        return (MWXFail);
    }
    MWX_Printf("(%d, %d), (%d, %d)\n", x11, y11, x22, y22);
    return (MWXSc_Make_ListSCM(4, rtn));
}

void scmTestInitFunc()
{
    MWX_Debug();
    MWXSc_Register_Function ("myGetInputLine", myGetInputLine);
    MWXSc_Register_Function ("myOutputLine", myOutputLine);
    MWX_Printf ("scmTestInitDone\n");
}

```

MWXDb_Set_GetPointCallBack

Registers callback function for getting an input point from a mouse click.

Syntax

```

int MWXDb_Set_GetPointCallBack (
    int (*proc) (int *, int *)
)

```

Parameters

`proc` Callback function.

Return

`MWXSucces` Occurs if the callback function is successfully registered.

`MWXFail` Occurs if the call back function is not successfully registered.

Example

```
static int x1;
static int y1;
static int validate = 0;

int CheckPoint (
    int* x, int* y)
{
    if (( x!= NULL) && (y!= NULL)) {
        validate = 1;
        x1 = *x;
        y1 = *y;
    }
    else validate = 0;
    return 1;
}

/*****
=
= Test Get Input Point from mouse click
=
=*****/

MWXSCM_t myGetInputPoint (MWXSCM_t list)
{
    char          shape[256];
    int32 rc;
    MWXSCM_t      rtn[2];

    if (MWXSc_Scanf(list, "s",shape) != 1) {
        MWX_Printf("Syntax : myGetInputPoint \"shape\" \"n\");
        return (MWXFail);
    }

    MWXDb_Set_GetPointCallBack (CheckPoint);
    if( MWXDb_Get_InputData(shape) != MWXSucces) {
        MWX_Printf(" INFO : Can't get input point \"n\");
        return (MWXFail);
    }
}

/*****
=
```



```

= Test Output Point from mouse click
=
=*****/

MWXSCM_t myOutputPoint (MWXSCM_t list)
{
    MWXPoint_t    inputPoint;
    int32          rc;
    MWXSCM_t      rtn[2];

    MWX_Printf(" ***** Output point's coordinate ***** \n");
    if (validate) {
        rtn[0] = MWXSc_Make_IntegerSCM(x1);
        rtn[1] = MWXSc_Make_IntegerSCM(y1);
        validate = 0;
    }
    else {
        MWX_Printf(" INFO : Please get point first \n");
        return (MWXFail);
    }
    return (MWXSc_Make_ListSCM(2,rtn));
}

void scmTestInitFunc()
{
    MWX_Debug();
    MWXSc_Register_Function ("myGetInputPoint", myGetInputPoint);
    MWXSc_Register_Function ("myOutputPoint", myOutputPoint);
    MWX_Printf ("scmTestInitDone\n");
}

```

MWXDb_Set_HierRefLibMode

This function sets the hierarchy reference library mode. The initial value is FALSE (0). When the same cell name exists in two or more reference libraries, you need to set the mode to TRUE before opening the child cell.

Syntax

```

int MWXDb_Set_HierRefLibMode(
int mode /* INPUT */
)

```

Parameters

mode	Initial is 0. The user needs to set the value to 1 when same cell name exists in two or more reference libraries.

Return

MWXSucces	Occurs if function successfully sets the hierRefLibMode.
MWXFail	Occurs if function fails to set the hierRefLibMode

Example

```
rc = MWXDb_Set_HierRefLibMode(&mode);
switch (rc) {
    case MWXSucces:
        printf("Successfully set the hierRefLibMode to %d\n", mode);
        break;
    case MWXFail:
        printf("Fail to set the hierRefLibMode\n");
        break;
    Default:
        printf("Unknown return code \n");
        break;
}
```

MWXDb_Set_IOPad

Specifies the cell instance as an IOPad cell, if it is not a standard cell.

Syntax

```
int MWXDb_Set_IOPad (
    MWXCellId_t cellId /* INPUT */
)
```

Parameters

cellId Database object ID of the cell to which the cell instances belong.

Return

MWXFail Occurs if IOPad is not set successfully.

MWXSuccess Occurs if IOPad is set successfully.

Example

```
rc = MWXDb_Set_IOPad(cellId);
switch (rc) {
case MWXSuccess:
return (MWXSuccess);
case MWXFail:
MWX_Printf("ERROR -- Failed to SetIOPad\n");
return (MWXFail);
default:
MWX_Printf("ERROR -- unknown return code from
           SetIOPad\n");
return (MWXFail);
}
```

Scheme Equivalent

dbSetIOPad

(see Physical Implementation Online Help)

MWXDb_Set_NetSubType

Sets or specifies the netSubType of the specific net.

Syntax

```
int MWXDb_Set_NetSubType (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t netId, /* INPUT */
    char *netSubType /* INPUT */
)
```

Parameters

<code>cellId</code>	Database object ID of the cell to which the information applies.
<code>netId</code>	Specifies the database identifier of the net to which the information applies.
<code>netSubType</code>	Net subtype that applies to the net. Valid values are <code>corepadcore And PadCTS NetCTS RootNetnonCTSNet</code>

Return

<code>MWXFail</code>	Occurs if the net subtype for the net is successfully set.
<code>MWXSuccess</code>	Occurs if the net subtype for the net is not successfully set.

Example

```
if(MWXDb_Set_NetSubType(cellId,netId,netSubType)!= MWXSuccess) {  
    MWX_Printf("Fail to set net sub type \n");  
    return (MWXFail);  
}  
return (MWXSuccess);
```

MWXDb_Set_NetSubType_ByName

Sets a subtype for the net specified by its name.

Syntax

```
int MWXDb_Set_NetSubType_ByName (  
    MWXCellId_t cellId, /* INPUT */  
    char *netName, /* INPUT */  
    char *netSubType /* INPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell.
netName	Specifies the net name.
netSubType	Specifies the net subtype. Valid values are corepadcore And padCTSNetCTS Root NetnonCTSNet

Return

MWXFail	Occurs if the subtype cannot be successfully set for the net.
MWXSuccess	Occurs if the subtype is successfully set for the net.

Example

```
rc = MWXDb_Set_NetSubTypeByName ( cellId, netId, netSubType );
switch {
case MWXSuccess:
return (MWXSuccess)
case MWXFail:
MWX_Printf("ERROR -- Failed to operate. \n");
return (MWXFail);
}
```

Scheme Equivalent

dbSetNetSubTypeByName
(see Physical Implementation Online Help)

MWXDb_Set_NetType

Sets the type of the net to signal, clock, power, or ground.

Syntax

```
int MWXDb_Set_NetType(
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t netId, /* INPUT */
    char *netType /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell containing the net.
netId	Specifies the database identifier of the net for which the type will be changed.
netType	Specifies the net type Milkyway assigns to the net (signal, clock, power, or ground). Note: If the design is eventually going to be used in IC Compiler, the net type can be changed from signal to clock or vice versa. It can also be changed from power to ground or vice versa. The net type cannot be changed from signal or clock to power or ground.

Return

MWXSuccess	Occurs if the net type is successfully set.
MWXFail	Occurs if the net type is not successfully set.

Example

```
MWXCellId_t cellId;
MWXObjId_t netId;
char netType[256];
int rc;
rc = MWXDb_Set_NetType (cellId, netId, netType);
switch (rc) {
case MWXSuccess:
return (MWXSuccess);
case MWXFail:
MWX_Printf("ERROR -- Failed to set net type\n");
return (MWXFail);
}
```

Scheme Equivalent

dbSetNetType
(see Physical Implementation Online Help)

MWXDb_Set_NetType_ByName

Sets the type of the net to signal, clock, power, or ground.

Syntax

```
int MWXDb_Set_NetType_ByName (
    MWXCellId_t cellId, /* INPUT */
    char *netName, /* INPUT */
    char *netType /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that owns the net to update.
netName	Specifies the name of the net to update.
netType	Specifies the type to assign to the net (valid values are signal, clock, power, or ground).

Note:

If the design is eventually going to be used in IC Compiler, the net type can be changed from signal to clock or vice versa. It can also be changed from power to ground or vice versa. The net type cannot be changed from signal or clock to power or ground.

Return

MWXSuccess	Occurs if the net type is successfully modified.
MWXFail	Occurs if the net type modification fails.

Example

```
rc = MWXDb_Set_NetType_ByName (cellId, netName, netType);
if (rc == MWXSuccess) {
    return (MWXSuccess);
} else {
    return (MWXFail);
}
```

Scheme Equivalent

dbSetNetTypeByName

(see Physical Implementation Online Help)

MWXDb_Set_RefControlFileMode

This function sets the reference control file mode. This function is needed when you dump, set, clear, or check the reference library control mode.

Syntax

```
int MWXDb_Set_RefControlFileMode(  
int mode /* INPUT */  
)
```

Parameters

mode	Reference control file mode (need to turn on before dump set clear or check the reference library control.

Return

MWXSuccess	Occurs if successfully sets the RefControlFileMode.
MWXFail	Occurs if fail to set the RefControlFileMode.

Example

```
    rc = MWXDb_Set_RefControlFileMode(1);  
switch (rc) {  
    case MWXSuccess:  
        printf("Successfully to set the reference control file mode");  
        break;  
    case MWXFail:  
        printf("Fail to set the reference control file mode\n");  
        break;  
    Default:  
        printf("Unknown return code \n");  
        break;  
}
```

MWXDb_Set_RefLibControl

This function sets the names of the reference library control from the input file. The Reference Library Control file mode must be turn on before using this function.

Syntax

```
int MWXDb_Set_RefLibControl(  
char *libName, /* INPUT */  
char *inputFile /* INPUT */  
)
```


Parameters

libName	Name of the library that you want to set the reference lib control.
inputFile	Name of the input file that contains all the names of the reference library control

Return

MWXSuccess	Occurs if Milkyway successfully sets the reference library control.
MWXFail	Occurs if Milkyway fails to set the reference library control.

Example

```
        Default:
        printf("Unknown return code \n");
        break;

}
```

MWXDb_Set_StdCell

Marks all instances of the specified cell to be StdCell.

Syntax

```
int MWXDb_Set_StdCell (
    MWXCellId_t cellId/* INPUT */
)
```

Parameters

cellId	Database object ID of the cell to which the cell instances belong.

Return

MWXFail	Occurs if StdCell is not successfully set.
MWXSuccess	Occurs if StdCell is successfully set.

Example

```
rc = MWXDb_Set_StdCell(cellId);
```

```

switch (rc) {
    case MWXSuccess:
        return (MWXSuccess);
    case MWXFail:
        return (MWXFail);
    default:
        return (MWXFail);
}

```

Scheme Equivalent

dbSetStdCell

(see Physical Implementation Online Help)

MWXDb_Set_TdfLoadCapacitance

Sets the load capacitance for the specified port.

Syntax

```

int32 MWXDb_Set_TdfLoadCapacitance (
    MWXCellId_t cellId, /* INPUT */
    char *portName, /* INPUT */
    int32 value /* INPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell.
portName	Specifies the port's name.
value	Specifies the load capacitance.

Return

MWXFail	Occurs if the load capacitance is not successfully set.
MWXSuccess	Occurs if the load capacitance is successfully set.

Example

```

rc = MWXDb_Set_TdfLoadCapacitance ( cellId, portName, value );
switch {
    case MWXSuccess:
        return (MWXSuccess)
    case MWXFail:
        MWX_Printf("ERROR -- Failed to operate. \n");
        return (MWXFail);
}

```

```
}
```

MWXdB_Set_TdfLoadCapacitanceInDouble

Sets the load capacitance for the specified port.

Syntax

```
int32 MWXdB_Set_TdfLoadCapacitanceInDouble (  
    MWXCellId_t cellId, /* INPUT */  
    char *portName, /* INPUT */  
    double value /* INPUT */  
)
```

Parameters

cellId	Specifies the cell's database identifier.
portName	Specifies the port's name.
value	Specifies the load capacitance.

Return

MWXFail	Occurs if the load capacitance is not successfully set.
MWXSuccess	Occurs if the load capacitance is successfully set.

Example

```
rc = MWXdB_Set_TdfLoadCapacitanceInDouble ( cellId, portName, value );  
switch {  
    case MWXSuccess:  
        return (MWXSuccess)  
    case MWXFail:  
        MWX_Printf("ERROR -- Failed to operate. \n");  
        return (MWXFail);  
}
```

MWXdB_Set_TdfSrcCapacitanceList

Sets the source capacitance list for the specified port.

Syntax

```
int32 MWXdB_Set_TdfSrcCapacitanceList (  
    MWXCellId_t cellId, /* INPUT */  
    char *portName, /* INPUT */
```

MWXdB_Set_TdfLoadCapacitanceInDouble

```

        int32 values [] /* INPUT */
    )

```

Parameters

cellId	Specifies the database identifier of the cell.
portName	Specifies the name of the port.
values []	Specifies the list of source capacitance values.

Return

MWXFail	Occurs if the source capacitance list is not successfully set.
MWXSuccess	Occurs if the source capacitance list is successfully set.

Example

```

rc = MWXDb_Set_TdfSrcCapacitanceList ( cellId, portName, values );
switch {
    case MWXSuccess:
        return (MWXSuccess)
    case MWXFail:
        MWX_Printf("ERROR -- Failed to operate. \n");
        return (MWXFail);
}

```

MWXDb_Set_TdfSrcCapacitanceListInDouble

Sets the source capacitance list for the specified port.

Syntax

```

int32 MWXDb_Set_TdfSrcCapacitanceListInDouble (
    MWXCellId_t cellId, /* INPUT */
    char *portName, /* INPUT */
    double values [] /* INPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell.
portName	Specifies the name of the port.
values []	Specifies the list of source capacitance values.

Return

MWXFail	Occurs if the source capacitance list is not successfully set.
MWXSuccess	Occurs if the source capacitance list is successfully set.

Example

```
rc = MWXDb_Set_TdfSrcCapacitanceListInDouble ( cellId, portName, values );
switch {
    case MWXSuccess:
        return (MWXSuccess)
    case MWXFail:
        MWX_Printf("ERROR -- Failed to operate. \n");
        return (MWXFail);
}
```

MWXDb_Set_TdfSrcResistanceList

Sets the source resistance list for the specified port.

Syntax

```
int32 MWXDb_Set_TdfSrcResistanceList (
    MWXCellId_t cellId, /* INPUT */
    char *portName, /* INPUT */
    int32 values [] /* INPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell.
<code>portName</code>	Specifies the name of the port.
<code>values []</code>	Specifies the list of source resistance values.

Return

<code>MWXFail</code>	Occurs if the source resistance list is not successfully set.
<code>MWXSuccess</code>	Occurs if the source resistance list is successfully set.

Example

```
rc = MWXDb_Set_TdfSrcResistanceList ( cellId, portName, values );

switch {
case MWXSuccess:
    return (MWXSuccess)
case MWXFail:
    MWX_Printf("ERROR -- Failed to operate. \n");
    return (MWXFail);
}
```

MWXDb_Set_TdfSrcResistanceListInDouble

Sets source resistance list for the specified port.

Syntax

```
int32 MWXDb_Set_TdfSrcResistanceListInDouble (
    MWXCellId_t cellId, /* INPUT */
    char *portName, /* INPUT */
    double values [] /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell.
portName	Specifies the name of the port.
values []	Specifies the list of source resistance values.

Return

MWXFail	Occurs if the source resistance list is not successfully set.
MWXSuccess	Occurs if the source resistance list is successfully set.

Example

```
rc = MWXDb_Set_TdfSrcResistanceListInDouble ( cellId, portName, values );
switch {
    case MWXSuccess:
        return (MWXSuccess)
    case MWXFail:
        MWX_Printf("ERROR -- Failed to operate. \n");
        return (MWXFail);
}
```

MWXDb_Set_TechFileName

Replaces the existing technology file for an open library. You *must* close and reopen the library.

Syntax

```
int MWXDb_Set_TechFileName (
    char *techFileName /* INPUT */
)
```

Parameters

techFileName	Name of the technology file that replaces the current library's existing technology file.

Return

MWXNoLibraryOpened	Occurs if library is not open.
MWXFail	Occurs if the technology file is not successfully replaced.
MWXSuccess	Occurs if the technology file is successfully replaced.

Example

```
rc = MWXDb_Set_TechFileName(fileName);
switch (rc) {
    case MWXSuccess:
        return (MWXSuccess);
    case MWXFail:
        return (MWXFail);
    case MWXNoLibraryOpened:
        return (MWXSc_Make_IntegerSCM(rc));
    default:
        return (MWXFail);
}
```

Scheme Equivalent

dbSetTechFileName
(see Physical Implementation Online Help)

MWXDb_Show_DiodePortConn

Displays the information for the cellInst, port, and net of the Diode PortInst that are contained in the cell identified by cellId.

Syntax

```
int MWXDb_Show_DiodePortConn(
    MWXCellId_t cellId /* INPUT : cell identification
                        number. */
    MWXObjId_t portInstId /* INPUT : port instance
                           identification number. */
)
```


Parameters

cellId	Specifies the database identifier of the cell containing Diode PortInstMasters.
portInstId	Specifies the database identifier of the port instance.

Return

MWXFail	Occurs when the information is successfully displayed.
MWXSuccess	Occurs when the information is not successfully displayed.

Example

```
rc = MWXDb_Show_DiodePortConn(cellId, portInstId);
    switch (rc) {
        case MWXSuccess:
            return (MWXSuccess);
        case MWXFail:
            MWX_Printf("Fail to display the information
                        for cellInst, port, and net of the Diode
                        PortInst.\n");
            return (MWXFail);
        default:
            return (MWXFail);
    }
```

MWXDb_Startup_ShapeStruct

This command prepares the shape data structure of a specified cell for faster geometry queries. This function should be called prior to making any attempt to initiate a sequence of object or instance searches by a bounding box. It is your responsibility to call [MWXDb_Cleanup_ShapeStruct](#) to recover memory that has been allocated for the shape data structure.

Note:

This function should be used only in association with [MWXDb_Get_Objects_ByBBoxSearch](#) and [MWXDb_Get_Instances_ByBBoxSearch](#).

Syntax

```
int MWXDb_Startup_ShapeStruct(
MWXCellId_t cellId /* INPUT */
)
```

Parameters

`cellId` Specifies the database identifier of the cell for which to prepare the shape data structure.

Return

`MWXSuccess` Occurs when the shape data structure has been successfully created.

`MWXFail` Occurs when the shape data structure is not successfully created for any reason.

Example

```
rc = MWXDb_Startup_ShapeStruct (cellId);
switch (rc) {
    case MWXSuccess:
        return (MWXSuccess);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to release shape data structure\n");
        return (MWXFail);
}
```

Scheme Equivalent

`dbInitShapeStruct`
(see Physical Implementation Online Help)

See Also

[MWXDb_Cleanup_ShapeStruct](#)
[MWXDb_Get_Instances_ByBBoxSearch](#)
[MWXDb_Get_Objects_ByBBoxSearch](#)

MWXDb_Store_CellAttachFile

Moves a UNIX file, `appFileName`, into the opened library and attaches the file to the specified cell as `pseudoAccessName`.

Syntax

```
int MWXDb_Store_CellAttachFile (
    MWXCellId_t cellId, /* INPUT */
    char *appFileName, /* INPUT */
    char *pseudoAccessName /* INPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell to which the UNIX file will be attached.
<code>appFileName</code>	Name of the UNIX file to be attached to the cell.
<code>pseudoAccessName</code>	Name to be assigned to the attached file for future access.

Return

<code>MWXNoLibraryOpened</code>	Occurs if no library is open.
<code>MWXFail</code>	Occurs if the file cannot be moved and successfully attached to the cell.
<code>MWXSuccess</code>	Occurs if the file is successfully attached.

Example

```
rc = MWXDb_Store_CellAttachFile ( cellId,fileName );
switch (rc) {
    case MWXSuccess:
        return (MWXSuccess);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to store CellAttachFile\n");
        return (MWXFail);
    case MWXNoLibraryOpened:
        MWX_Printf("ERROR -- No open library\n");
        return (MWXFail);
    default:
        MWX_Printf("ERROR -- unknown return code from
        MWXDb_Store_CellAttachFile\n");
        return (MWXFail);
}
```

Scheme Equivalent

`dbStoreCellAttachedFile`
(see Physical Implementation Online Help)

MWXDb_Store_LibAttachFile

Moves a UNIX file, `appFileName`, into the opened library and attaches the file to the open library as `pseudoAccessName`.

Syntax

```
int MWXDb_Store_LibAttachFile (
    char *appFileName, /* INPUT */
    char *pseudoAccessName /* INPUT */
)
```

Parameters

appFileName	Name of the UNIX file to be attached to the library.
pseudoAccessName	Name to be assigned to the attached file for future access.

Return

MWXNoLibraryOpened	Occurs if no library is open.
MWXFail	Occurs if the file cannot be moved and successfully attached to the library.
MWXSuccess	Occurs if the file is successfully attached.

Example

```
rc = MWXDb_Store_CellAttachFile ( fileName );
switch (rc) {
    case MWXSuccess:
        return (MWXSuccess);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to store libAttachFile\n");
        return (MWXFail);
    case MWXNoLibraryOpened:
        MWX_Printf("ERROR -- No open library\n");
        return (MWXFail);
    default:
        MWX_Printf("ERROR -- unknown return code from
        MWXDb_Store_LibAttachFile\n");
        return (MWXFail);
}
```

Scheme Equivalent

dbStoreLibAttachedFile
(see Physical Implementation Online Help)

MWXDb_Transform_Object

Rotates and flips a specified object.

Syntax

```
int MWXDb_Transform_Object(  
    MWXCellId_t cellId,      /* INPUT */  
    MWXObjId_t objId,       /* INPUT */  
    char *rotationStr,      /* INPUT */  
    char *mirrorStr,        /* INPUT */  
    MWXPoint_t *pivot,      /* INPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell that was opened.
objId	Specifies the database identifier of the object that needs transformation.
rotationStr	Specifies the counterclockwise rotation value, in degrees (0, 90, 180, 270).
mirrorStr	Specifies the direction, if any, in which to flip the cell instance (no, X or Y).
pivot	Specifies the x/y-coordinates representing the pivot for movement.

Return

MWXSuccess	Occurs when the object is successfully transformed.
MWXFail	Occurs when the object is not successfully transformed.

Example

```
    pivot.x = (int32) x ;  
    pivot.y = (int32) y ;  
    rc = MWXDb_Transform_Object (cellId, objectId, "90", "X",  
        &pivot);  
    switch (rc) {  
        case MWXSuccess:  
            return (MWXSuccess);  
        case MWXFail:  
            MWX_Print("ERROR -- Failed to transform object\n");  
            return (MWXFail);  
    }
```

MWXDb_Truncate_CellText

Removes all the characters appearing after the first occurrence of the character in startChar from all the text objects appearing in a cell. Truncation includes the character in startChar.

Syntax

```
int MWXDb_Truncate_CellText (  
    MWXCellId_t cellId, /* INPUT */  
    char *startChar /* INPUT */  
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell to change.
<code>startChar</code>	Specifies the character after which all the characters are removed.

Return

<code>MWXSuccess</code>	Occurs if the truncation is successful.
<code>MWXFail</code>	Occurs if the truncation fails.

Example

```
rc = MWXDb_Truncate_CellText (cellId, startChar);  
if (rc != MWXSuccess) {  
    return (MWXFail);  
} else {  
    return (MWXSuccess);  
}
```

Scheme Equivalent

```
dbTruncateCellText  
(see Physical Implementation Online Help)
```

MWXDb_Truncate_Text

Removes all the characters appearing after the first occurrence of the character in `startChar` from all text objects appearing in all the cells of a library. Truncation includes the character in `startChar`.

Syntax

```
int MWXDb_Truncate_Text (  
    MWXLibId_t libId, /* INPUT */  
    char *startChar /* INPUT */  
)
```

Parameters

<code>libId</code>	Specifies the database identifier of the library containing the cells to change.
<code>startChar</code>	Specifies the character after which all the characters are removed.

Return

<code>MWXSuccess</code>	Occurs if the truncation is successful.
<code>MWXFail</code>	Occurs if the truncation fails.

Example

```
rc = MWXDb_Truncate_Text (libId, startChar);
if (rc != MWXSuccess) {
    return (MWXFail);
} else {
    return (MWXSuccess);
}
```

Scheme Equivalent

`dbTruncateText`

(see Physical Implementation Online Help)

MWXDb_Update_CellProperty_ByName

Changes the value of an existing property associated with a cell. The cell must be open and a property with the name `propertyName` must exist. The `propertyId` will be undefined if the return value is not `MWXSuccess`.

Syntax

```
int MWXDb_Update_CellProperty_ByName (
    MWXCellId_t cellId, /* INPUT */
    char *propertyName, /* INPUT */
    unsigned char propertyDataType, /* INPUT */
    MWXPropValue_t *propertyValue, /* INPUT */
    uint16 propertySize, /* INPUT */
    MWXObjId_t *propertyId /* OUTPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that owns the property being changed.
propertyName	Specifies the name of the property to change.
propertyDataType	Specifies the new data type to associate with the property. Valid values are MWXcPropBooleanTypeMWXcPropIntegerTypeMWXcPropFloatingTypeMWXcPropStringTypeMWXcPropDataStructTypeMWXcPropTimeType
propertyValue	Specifies the new value to assign to the property.
propertySize	Specifies the memory size of the property value (for string, use string length; for all others, use the size of the data type).
propertyId	Specifies the database identifier of the updated property.

Return

MWXSuccess	Occurs if the property is successfully updated.
MWXFail	Occurs if the property update fails.

Example

```
propertyValue.pString = config_file;
rc = MWXDb_Update_CellProperty_ByName (cellId, "ConfigFile",
MWXcPropStringType, propertyValue, (uint16) strlen (propertyValue.pString),
&propertyId);
if (rc == MWXSuccess) {
    return (MWXSc_Make_DBIIdSCM (propertyId));
} else {
    return (MWXFail);
}
```

Scheme Equivalent

dbUpdateCellProperty
(see Physical Implementation Online Help)

MWXDb_Update_Property

Changes the value of an existing property associated with an object. The cell that owns the object must be open, and a property identified by `propertyId` must exist. Note that `propertyId` will be undefined if the return value is not `MWXSuccess`.

Syntax

```
int MWXDb_Update_Property (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t propertyId, /* INPUT */
    unsigned char propertyDataType, /* INPUT */
    MWXPropValue_t *propertyValue, /* INPUT */
    uint16 propertySize /* INPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell that owns the object that owns the property being changed.
<code>propertyId</code>	Specifies the database identifier of the property to change.
<code>propertyDataType</code>	Specifies a new data type to associate with the property. Valid values are <code>MWXcPropBooleanType</code> <code>MWXcPropIntegerType</code> <code>MWXcPropFloatingType</code> <code>MWXcPropStringType</code> <code>MWXcPropDataStructType</code> <code>MWXcPropTimeType</code>
<code>propertyValue</code>	Specifies a new value to assign to the property.
<code>propertySize</code>	Specifies the memory size of the property value (for string, use string length; for all others, use the size of the data type).

Return

<code>MWXSuccess</code>	Occurs if the property is successfully updated.
<code>MWXFail</code>	Occurs if the property update fails.

Example

```
rc = MWXDb_Update_Property (cellId, propertyId, MWXcPropStringType,
propertyValue, (uint16) strlen (propertyValue.pString));
if (rc == MWXSuccess) {
    return (MWXSuccess);
} else {
    return (MWXFail);
}
```

Scheme Equivalent

`dbUpdateProperty`
(see Physical Implementation Online Help)

MWXDb_Update_Property_ByName

Changes the value of an existing property associated with a cell. The cell that owns the object must be open, and a property with the name `propertyName` must exist. Note that `propertyId` will be undefined if the return value is not `MWXSuccess`.

Syntax

```
int MWXDb_Update_Property_ByName (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t objectId, /* INPUT */
    char *propertyName, /* INPUT */
    unsigned char propertyDataType, /* INPUT */
    MWXPropValue_t *propertyValue, /* INPUT */
    uint16 propertySize, /* INPUT */
    MWXObjId_t *propertyId /* OUTPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell that owns the object identified by <code>objectId</code> being changed.
<code>objectId</code>	Specifies the database identifier of the object that owns the property being changed.
<code>propertyName</code>	Specifies the name of the property to change.
<code>propertyDataType</code>	Specifies the new data type to associate with the property. Valid values are <code>MWXCPropBooleanType</code> <code>MWXCPropIntegerType</code> <code>MWXCPropFloatingType</code> <code>MWXCPropStringType</code> <code>MWXCPropDataStructType</code> <code>MWXCPropTimeType</code>
<code>propertyValue</code>	Specifies a new value to assign to the property.
<code>propertySize</code>	Specifies a memory size of the property value (for string, use string length; for all others, use the size of the data type).
<code>propertyId</code>	Specifies the database identifier of the updated property.

Return

<code>MWXSuccess</code>	Occurs if the property is successfully updated.
<code>MWXFail</code>	Occurs if the property update fails.

Example

```
propertyValue.pString = config_file;
rc = MWXDb_Update_Property_ByName (cellId, objectId, "ConfigFile",
MWXCPropStringType, propertyValue, (uint16) strlen (propertyValue.pString),
&propertyId);
if (rc == MWXSuccess) {
    return (MWXSc_Make_DBIIdSCM (propertyId));
} else {
    return (MWXFail);
}
```

5

Technology File Data Access Functions

This chapter lists the `MWXdB_Get *` functions, which are data access functions you can use in a technology file.

For more information on technology file support, see the *Milkyway Environment Data Preparation User Guide*.

MWXdB_Get_ColorName

The `MWXdB_Get_ColorName` function gets the ColorRGB name from the technology file by the given color ID.

Syntax

```
int MWXdB_Get_ColorName(  
    int32 **colorId  
    char **colorName  
)
```

Parameters

<code>colorId</code>	Specifies the database identifier of the color from the technology file.
<code>colorName</code>	Name of the colorRGB.

Return

<code>MWXSuccess</code>	Occurs when you successfully get the colorName.
<code>MWXFail</code>	Occurs when you fail to successfully get the colorName.

Example

```
rc = MWXdB_Get_ColorName (colorId,&colorname);  
switch (rc) {  
    case MWXSuccess:  
        MWX_Printf ("colorName=%s\n",colorName) ;  
        return (MXW_TRUE_SCM)  
    case MWXFail:  
        MWX_Printf("ERROR -- failed to get color name\n");  
        return (MWXFail);  
    case MWXNoLibraryOpened:  
        MWX_Printf ("ERROR -- No library open\n") ;  
        return (MWXFail);  
default:  
        MWX_Printf ("ERROR -- unknown return code from  
MWXdB_Get_ColorName\n") ;  
        return (MWXFail) :}
```

MWXdB_Get_ColorRGB

The `MWXdB_Get_ColorRGB` function gets the intensity of red, green, and blue of the colorRGB from the given color ID.

Syntax

```
int MWXDb_Get_ColorRGB(  
    int32 colorId,  
    int32 *redIntensity  
    int32 *greenIntensity,  
    int32 *blueIntensity  
)
```

Parameters

colorId	Specifies the database identifier of the color from the technology file.
redIntensity	The intensity of the color red in the given colorId.
greenIntensity	The intensity of the color green in the given colorId.
blueIntensity	The intensity of the color blue in the given colorId.

Return

MWXSuccess	Indicates a successful operation.
MWXFail	Indicates that the operation failed.

Example

```
rc = MWXDb_Get_ColorRGB (colorId, &red, &green, &blue);  
switch (rc) {  
    case MWXSuccess:  
        MWX_Printf ("Red=%d, Green=%d, Blue=%d\n", red,green,blue) ;  
        return (MXW_TRUE_SCM)  
    case MWXFail:  
        MWX_Printf("ERROR -- failed to get color name\n");  
        return (MWXFail);  
    case MWXNoLibraryOpened:  
        MWX_Printf ("ERROR -- No library open\n") ;  
        return (MWXFail);  
  
    default:  
        MWX_Printf ("ERROR -- unknown return code from  
MWXDb_Get_ColorRGB\n") ;  
        return (MWXFail) ;  
}
```

MWXDb_Get_ColorRGBDefined

The MWXDb_Get_ColorRGBDefined function gets the ColorRGB defined from the technology file by the given color ID.

Syntax

```
int MWXDb_Get_ColorName(  
    int32 **colorName
```

Parameters

colorId	Specifies the database identifier of the color from the technology file.
rgbDefined	Is 1 if the color is defined as a colorRGB in the technology file.

Return

MWXSuccess	Occurs when you successfully get colorRGBDefined.
MWXFail	Occurs when you fail to successfully get the colorRGBDefined.

Example

```
rc = MWXDb_Get_ColorRGBDefined (colorId, &rgbDefined);  
switch (rc) {  
    case MWXSuccess:  
        MWX_Printf ("colorRGBDefined=%d\n",rgbDefined) ;  
        return (MXW_TRUE_SCM)  
    case MWXFail:  
        MWX_Printf("ERROR -- failed to get colorRGBDefined\n");  
        return (MWXFail);  
    case MWXNoLibraryOpened:  
        MWX_Printf ("ERROR -- No library open\n") ;  
        return (MWXFail) ;  
default:  
        MWX_Printf ("ERROR -- unknown return code \n") ;  
        return (MWXFail) :}
```

MWXDb_Get_DesignRuleDiffNetMinSpacing

The MWXDb_Get_DesignRuleDiffNetMinSpacing function retrieves the design rule of the diff net min spacing between two given layers in the technology file.

Syntax

```
public int32 MWXDb_Get_DesignRuleDiffNetMinSpacing(  
    int32 layerNumber1, /* INPUT */  
    int32 layerNumber2, /* INPUT */  
    double *diffNetMinSpacing /* OUTPUT */  
)
```


Parameters

layerNumber1	The first layer from which the diff net min spacing in the design rule is being retrieved.
layerNumber2	The second layer from which the diff net min spacing in the design rule is being retrieved.
diffNetMinSpacing	The different net of the min spacing in the design rule between the two given layers.

Return

MWXSuccess	Occurs if Milkyway successfully retrieves the design rule of the diff net min spacing.
MWXFail	Occurs if Milkyway fails to successfully retrieve the design rule of the diff net min spacing.

Example

```
MWX_Printf("INFO -- Getting designRule diff Net minSpacing...\n");
rc = MWXDb_Get_DesignRuleDiffNetMinSpacing (layer1, layer2, &diffNetMinSpacing);
switch (rc) {
case MWXSuccess:
return (MWXSc_Make_DoubleSCM(diffNetMinSpacing));
case MWXFail:
MWX_Printf("ERROR -- Failed to get design rule
diffNetMinSpacing\n");
return (MWXFail);
default:
MWX_Printf("ERROR -- unknown return code from
MWXDb_Get_DesignRule\n");
return (MWXFail);
}
```

MWXDb_Get_DesignRuleMinEnclosure

The MWXDb_Get_DesignRuleMinEnclosure function retrieves the design rule of the minimum enclosure between two given layers in the technology file.

Syntax

```
public int32 MWXDb_Get_DesignRuleMinEnclosure(
    int32 layerNumber1, /* INPUT */
    int32 layerNumber2, /* INPUT */
    double *minEnclosure /* OUTPUT */
)
```

Parameters

layerNumber1	The first layer from which the minimum enclosure in the design rule is being retrieved.
layerNumber2	The second layer from which the minimum enclosure in the design rule is being retrieved.
minEnclosure	The minimum enclosure in the designRule between the two given layers.

Return

MWXSuccess	Occurs if Milkyway successfully retrieves the design rule of the minimum enclosure.
MWXFail	Occurs if Milkyway fails to successfully retrieve the design rule of the minimum enclosure.

Example

```
MWX_Printf("INFO -- Getting designRule minEnclosure...\n");
rc = MWXDb_Get_DesignRuleMinEnclosure (layer1, layer2, &minEnclosure);
    switch (rc) {
    case MWXSuccess:
        return (MWXSc_Make_DoubleSCM(minEnclosure));
    case MWXFail:
        MWX_Printf("ERROR -- Failed to get design rule minEnclosure\n");
        return (MWXFail);
    default:
        MWX_Printf("ERROR -- unknown return code from MWXDb_Get_DesignRule\n");
    }
return (MWXFail);
}
```

MWXDb_Get_DesignRuleMinSpacing

The MWXDb_Get_DesignRuleMinSpacing function retrieves the design rule of the minimum spacing between two given layer in the technology file.

Syntax

```
public int32 MWXDb_Get_DesignRuleMinSpacing(
    int32 layerNumber1, /* INPUT */
    int32 layerNumber2, /* INPUT */
    double *minSpacing /* OUTPUT */
)
```

Parameters

layerNumber1	The first layer from which the minimum spacing in the design rule is being retrieved.
layerNumber2	The second layer from which the minimum spacing in the design rule is being retrieved.
minSpacing	The designRule of the minimum spacing between the two given layers.

Return

MWXSuccess	Occurs if Milkyway successfully retrieves the design rule of the minimum spacing.
MWXFail	Occurs if Milkyway fails to successfully retrieve the design rule of the minimum spacing.

Example

```
MWX_Printf("INFO -- Getting designRule minSpacing...\n");
rc = MWXDb_Get_DesignRuleMinSpacing (layer1, layer2, &minSpacing);
switch (rc) {
    case MWXSuccess:
        return (MWXSc_Make_DoubleSCM(minSpacing));
    case MWXFail:
        MWX_Printf("ERROR -- Failed to get design rule minSpacing\n");
        return (MWXFail);
    default:
        MWX_Printf("ERROR -- unknown return code from MWXDb_Get_DesignRule\n");
        return (MWXFail);
}
```

MWXDb_Get_DesignRuleStackable

The MWXDb_Get_DesignRuleStackable function retrieves the design rule of the stackable between two given layers in the technology file.

Syntax

```
public int32 MWXDb_Get_DesignRuleStackable(
    int32 layerNumber1, /* INPUT */
    int32 layerNumber2, /* INPUT */
    int32 *stackable /* OUTPUT */
)
```

Parameters

layerNumber1	The first layer from which the design rule of the stackable is being retrieved.
layerNumber2	The second layer from which design rule of the stackable is being retrieved.
stackable	The design rule of the stackable between the two given layers.

Return

MWXSuccess	Occurs if Milkyway successfully retrieves the design rule of the stackable.
MWXFail	Occurs if Milkyway fails to successfully retrieve the design rule of the stackable.

Example

```
MWX_Printf("INFO -- Getting designRule stackable...\n");
    rc = MWXDb_Get_DesignRuleStackable (layer1, layer2,&stackable);
switch (rc) {
    case MWXSuccess:
        return (MWXSc_Make_DoubleSCM(stackable));
    case MWXFail:
        MWX_Printf("ERROR -- Failed to get design rule stackable\n");
        return (MWXFail);
    default:
        MWX_Printf("ERROR -- unknown return code from MWXDb_Get_DesignRule\n");
        return (MWXFail);
}
```

MWXDb_Get_FringeCapInfo

The MWXDb_Get_FringeCapInfo function retrieves the FringeCap information from the technology file.

Syntax

```
public int32 MWXDb_Get_FringeCapInfo(
    int32 layerNumber1, /* INPUT */
    int32 layerNumber2, /* INPUT */
    double *minFringeCap, /* OUTPUT */
    double *nomFringeCap, /* OUTPUT */
    double *maxFringeCap /* OUTPUT */
)
```

Parameters

layerNumber1	The first number of the layer number from which the fringe cap is retrieved.
layerNumber2	The second number of the layer number from which the fringe cap is retrieved.
minFringeCap	The minimum value of the fringe cap.
nomFringeCap	The normal value of the fringe cap.
maxFringeCap	The maximum value of the fringe cap.

Return

MWXSuccess	Occurs if Milkyway successfully retrieves the design rule of the FringeCap information.
MWXFail	Occurs if Milkyway fails to successfully retrieve the design rule of the FringeCap information.

Example

```
MWX_Printf("INFO -- Getting FringeCap Info ...\n");
rc = MWXDb_Get_FringeCapInfo (layer1, layer2, &minFringeCap,
&nomFringeCap, &maxFringeCap);
switch (rc) {
case MWXSuccess:
    rtn[0] = MWXSc_Make_DoubleSCM(minFringeCap);
    rtn[1] = MWXSc_Make_DoubleSCM(nomFringeCap);
    rtn[2] = MWXSc_Make_DoubleSCM(maxFringeCap);
    return (MWXSc_Make_ListSCM (3, rtn));
case MWXFail:
    MWX_Printf("ERROR -- Failed to get fringeCapInfo\n");
    return (MWXFail);
default:
    MWX_Printf("ERROR -- unknown return code from MWXDb_Get_FringeCapInfo\n"); return
    (MWXFail);
}
```

MWXDb_Get_LayerBorderStyleName

The MWXDb_Get_LayerBorderStyleName function gets the name of the layer border style (keyword `borderStyle` in the technology file) for a layer in a library.

Note:

There is no need to free `borderStyleName`.

Syntax

```
int MWXDb_Get_LayerBorderStyleName (  
    int32 layerId, /* INPUT */  
    char **borderStyleName /* OUTPUT */  
)
```

Parameters

layerId	Specifies the index into the database layer table.
borderStyleName	Specifies the name of the border style.

Return

MWXSuccess	Occurs if the data is successfully retrieved.
MWXNoLibraryOpened	Occurs if no library is open.
MWXFail	Occurs if the operation fails for any other reason.

Example

```
if (MWXDb_Get_LayerId_ByName ("metall", &layerId) == MWXSuccess) {  
    if (MWXDb_Get_LayerBorderStyleName (layerId,  
        &borderStyle) == MWXSuccess) {  
        rtn = MWXSc_Make_StringSCM (borderStyle);  
        return (rtn);  
    }  
}
```

MWXDb_Get_LayerCapacitance

The `MWXDb_Get_LayerCapacitance` function gets the capacitance values for a layer in a library. The associated keywords in the technology file are

- unitMinCapacitance
- unitNomCapacitance
- unitMaxCapacitance
- unitMinChannelCap
- unitNomChannelCap
- unitMaxChannelCap
- unitMinSideWallCap
- unitNomSideWallCap

- unitMaxSideWallCap
- unitMinChannelSideCap
- unitNomChannelSideCap
- unitMaxChannelSideCap

Syntax

```
int MWXDb_Get_LayerCapacitance (
    int32 layerId, /* INPUT */
    double *minCapacitance, /* OUTPUT */
    double *nomCapacitance, /* OUTPUT */
    double *maxCapacitance, /* OUTPUT */
    double *minChannelCapacitance, /* OUTPUT */
    double *nomChannelCapacitance, /* OUTPUT */
    double *maxChannelCapacitance, /* OUTPUT */
    double *minSideCapacitance, /* OUTPUT */
    double *nomSideCapacitance, /* OUTPUT */
    double *maxSideCapacitance, /* OUTPUT */
    double *minChannelSideCapacitance, /* OUTPUT */
    double *nomChannelSideCapacitance, /* OUTPUT */
    double *maxChannelSideCapacitance/* OUTPUT */
)
```

Parameters

<code>layerId</code>	Specifies the index into the database layer table.
<code>minCapacitance</code>	Specifies the minimum capacitance per square unit in a cell instance.
<code>nomCapacitance</code>	Specifies the nominal capacitance per square unit in a cell instance.
<code>maxCapacitance</code>	Specifies the maximum capacitance per square unit in a cell instance.
<code>minChannelCapacitance</code>	Specifies the minimum capacitance per square unit in a routing channel.
<code>nomChannelCapacitance</code>	Specifies the nominal capacitance per square unit in a routing channel.
<code>maxChannelCapacitance</code>	Specifies the maximum capacitance per square unit in a routing channel.
<code>minSideCapacitance</code>	Specifies the minimum sidewall capacitance per unit length in a cell instance.
<code>nomSideCapacitance</code>	Specifies the nominal sidewall capacitance per unit length in a cell instance.
<code>maxSideCapacitance</code>	Specifies the maximum sidewall capacitance per unit length in a cell instance.
<code>minChannelSideCapacitance</code>	Specifies the minimum sidewall capacitance per unit length in a routing channel.
<code>nomChannelSideCapacitance</code>	Specifies the nominal sidewall capacitance per unit length in a routing channel.
<code>maxChannelSideCapacitance</code>	Specifies the maximum sidewall capacitance per unit length in a routing channel.

Return

<code>MWXSuccess</code>	Occurs if the data is successfully retrieved.
<code>MWXNoLibraryOpened</code>	Occurs if no library is open.
<code>MWXFail</code>	Occurs if the operation fails for any other reason.

Example

```
if (MWXDb_Get_LayerId_ByName ("metall", &layerId) == MWXSuccess) {
    if (MWXDb_Get_LayerCapacitance (layerId,
        &minCapacitance, &nomCapacitance, &maxCapacitance,
        &minChannelCapacitance, &nomChannelCapacitance,
        &maxChannelCapacitance,
        &minSideCapacitance, &nomSideCapacitance,
        &maxSideCapacitance,
        &minChannelSideCapacitance, &nomChannelSideCapacitance,
        &maxChannelSideCapacitance) == MWXSuccess) {
        rtn[0] = MWXSc_Make_DoubleSCM(minCapacitance);
        rtn[1] = MWXSc_Make_DoubleSCM(nomCapacitance);
        rtn[2] = MWXSc_Make_DoubleSCM(maxCapacitance);
        rtn[3] = MWXSc_Make_DoubleSCM(minChannelCapacitance);
        rtn[4] = MWXSc_Make_DoubleSCM(nomChannelCapacitance);
        rtn[5] = MWXSc_Make_DoubleSCM(maxChannelCapacitance);
        rtn[6] = MWXSc_Make_DoubleSCM(minSideCapacitance);
        rtn[7] = MWXSc_Make_DoubleSCM(nomSideCapacitance);
        rtn[8] = MWXSc_Make_DoubleSCM(maxSideCapacitance);
        rtn[9] = MWXSc_Make_DoubleSCM(minChannelSideCapacitance);
        rtn[10] = MWXSc_Make_DoubleSCM(nomChannelSideCapacitance);
        rtn[11] = MWXSc_Make_DoubleSCM(maxChannelSideCapacitance);
        return (MWXSc_Make_ListSCM (12, rtn));
    }
}
```

MWXDb_Get_LayerColorName

The MWXDb_Get_LayerColorName function gets the name of the layer color (keyword color in the technology file) for a layer in a library.

Syntax

```
int MWXDb_Get_LayerColorName (
    int32 layerId,                /* INPUT */
    char **colorName /* OUTPUT */
)
```

Parameters

layerId	Specifies the index into the database layer table.
colorName	Specifies the name of the color for the layer.

Return

MWXSucces	Occurs if the data is successfully retrieved.
MWXNoLibraryOpened	Occurs if no library is open.
MWXFail	Occurs if the operation fails for any other reason.

Example

```
if (MWXDb_Get_LayerId_ByName ("metall", &layerId) == MWXSucces) {
    if (MWXDb_Get_LayerColorName (layerId, &colorName)
        == MWXSucces) {
        rtn = MWXSc_Make_StringSCM (colorName);
        free (colorName);
        return (rtn);
    }
}
```

MWXDb_Get_LayerDisplayAttr

The MWXDb_Get_LayerDisplayAttr function gets the display attributes for a layer in a library. The associated keywords in the technology file are

- visible
- selectable
- blink
- outlineWidth
- panelNumber

Syntax

```
int MWXDb_Get_LayerDisplayAttr (
    int32 layerId, /* INPUT */
    uchar *visible, /* OUTPUT */
    uchar *selectable, /* OUTPUT */
    uchar *blink, /* OUTPUT */
    double *outlineWidth, /* OUTPUT */ ==>
    used to be in INT32 type
    uchar *panelNumber /* OUTPUT */
)
```

Parameters

layerId	Represents the index into the database layer table.
visible	1 if the layer is visible; 0 if the layer is not.
selectable	1 if the objects on the layer are selectable, 0 if not.
blink	1 if the layer blinks, 0 if not.
outlineWidth	Unused at this time.
panelNumber	Represents the subpanel for the layer (valid values are 0–3).

Return

MWXSucces	Occurs if the data is successfully retrieved.
MWXNoLibraryOpened	Occurs if no library is open.
MWXFail	Occurs if the operation fails for any other reason.

Example

```
if (MWXDb_Get_LayerId_ByName ("metall", &layerId) == MWXSucces) {
    if (MWXDb_Get_LayerDisplayAttr (layerId, &visible,
        &selectable, &blink, &outlineWidth, &panelNumber) ==
        MWXSucces) {
        rtn[0] = MWXSc_Make_BooleanSCM(visible);
        rtn[1] = MWXSc_Make_BooleanSCM(selectable);
        rtn[2] = MWXSc_Make_BooleanSCM(blink);
        rtn[3] = MWXSc_Make_IntegerSCM((int) outlineWidth);
        rtn[4] = MWXSc_Make_IntegerSCM((int) panelNumber);
        return (MWXSc_Make_ListSCM (5, rtn));
    }
}
```

MWXDb_Get_LayerEtchFactor

The MWXDb_Get_LayerEtchFactor function gets the etch factors for a layer in a library. The associated keywords in the technology file are

- minDeltaWidth
- nomDeltaWidth
- maxDeltaWidth

Syntax

```
int MWXDb_Get_LayerEtchFactor (
    int32 layerId, /* INPUT */
    double *minEtchFactor, /* OUTPUT *//====\
    double *nomEtchFactor, /* OUTPUT *//====\
        used to be in INT32 type
    double *maxEtchFactor /* OUTPUT *//====/
)
```

Parameters

layerId	Specifies the index into the database layer table.
minEtchFactor	Specifies the minimum distance by which objects may expand or shrink. The keyword for a layer in a technology file is minDeltaWidth.
nomEtchFactor	Specifies the nominal distance by which objects may expand or shrink. The keyword for a layer in a technology file is nomDeltaWidth.
maxEtchFactor	Specifies the maximum distance by which objects may expand or shrink. The keyword for a layer in a technology file is maxDeltaWidth.

Return

MWXSuccess	Occurs if the data is successfully retrieved.
MWXNoLibraryOpened	Occurs if no library is open.
MWXFail	Occurs if the operation fails for any other reason.

Example

```
if (MWXDb_Get_LayerId_ByName ("metall", &layerId) == MWXSuccess) {
    if (MWXDb_Get_LayerEtchFactor (layerId, &minEtchFactor,
        &nomEtchFactor, &maxEtchFactor) == MWXSuccess) {
        rtn[0] = MWXSc_Make_IntegerSCM(minEtchFactor);
        rtn[1] = MWXSc_Make_IntegerSCM(nomEtchFactor);
        rtn[2] = MWXSc_Make_IntegerSCM(maxEtchFactor);
        return (MWXSc_Make_ListSCM (3, rtn));
    }
}
```

MWXdB_Get_LayerFatWireRules

The MWXdB_Get_LayerFatWireRules function gets the fat wire factors for a layer in a library. The associated keywords in the technology file are

- fatWireThreshold
- fatWireContact
- fatThinMinSpacing
- fatFatMinSpacing

Syntax

```
int MWXdB_Get_LayerFatWireRules (
    int32 layerId, /* INPUT */
    double *fatWireThreshold, /* OUTPUT */
    double *fatContactThreshold, /* OUTPUT
        used to be in INT32 type */
    double *fatToThinWireSpacing, /* OUTPUT */
    double *fatToFatWireSpacing /* OUTPUT */
)
```

Parameters

layerId	Specifies the index into the database layer table.
fatWireThreshold	Width beyond which fat wire rules apply.
fatContactThreshold	Width beyond which a contact is required.
fatToThinWireSpacing	Minimum distance between a fat and nonfat wire.
fatToFatWireSpacing	Minimum distance between two fat wires.

Return

MWXSucces	Occurs if the data is successfully retrieved.
MWXNoLibraryOpened	Occurs if no library is open.
MWXFail	Occurs if the operation fails for any other reason.

Example

```
if (MWXdB_Get_LayerId_ByName ("metall", &layerId) == MWXSucces) {
    if (MWXdB_Get_LayerFatWireRules (layerId, &fatWireThreshold,
        &fatContactThreshold, &fatToThinWireSpacing,
        &fatToFatWireSpacing) == MWXSucces) {
        rtn[0] = MWXSc_Make_IntegerSCM(fatWireThreshold);
    }
}
```

```

    rtn[1] = MWXSc_Make_IntegerSCM(fatContactThreshold);
    rtn[2] = MWXSc_Make_IntegerSCM(fatToThinWireSpacing);
    rtn[3] = MWXSc_Make_IntegerSCM(fatToFatWireSpacing);
    return (MWXSc_Make_ListSCM (4, rtn));
}

```

MWXDb_Get_LayerFringeCap

The MWXDb_Get_LayerFringeCap function gets the fringe capacitance values for a layer in a library. The associated keywords in the technology file are defaultInterFringeCap and defaultIntraFringeCap.

Syntax

```

int MWXDb_Get_LayerFringeCap (
    int32 layerId, /* INPUT */
    double *intraLayerFringeCap, /* OUTPUT */
    double *interLayerFringeCap /* OUTPUT */
)

```

Parameters

layerId	Specifies the index into the database layer table.
intraLayerFringeCap	Specifies the value of defaultIntraFringeCap for the layer from the technology file.
interLayerFringeCap	Specifies the value of defaultInterFringeCap for the layer from the technology file.

Return

MWXSuccess	Occurs if the data is successfully retrieved.
MWXNoLibraryOpened	Occurs if no library is open.
MWXFail	Occurs if the operation fails for any other reason.

Example

```

if (MWXDb_Get_LayerId_ByName ("metall1", &layerId) == MWXSuccess)
{
    if (MWXDb_Get_LayerColorName MWXDb_Get_LayerFringeCap
        (layerId, &intraLayerFringeCap, &interLayerFringeCap)
        == MWXSuccess) {
        rtn[0] = MWXSc_Make_DoubleSCM(intraLayerFringeCap);
        rtn[1] = MWXSc_Make_DoubleSCM(interLayerFringeCap);
        return (MWXSc_Make_ListSCM (2, rtn));
    }
}

```

MWXdB_Get_LayerHeightFromSub

The MWXdB_Get_LayerHeightFromSub function gets the height-from-substrate values for a layer in a library. The associated keywords in the technology file are unitMinHeightFromSub, unitNomHeightFromSub, and unitMaxHeightFromSub.

Syntax

```
int MWXdB_Get_LayerHeightFromSub (  
    int32 layerId, /* INPUT */  
    double *minHeightFromSub, /* OUTPUT */  
    double *nomHeightFromSub, /* OUTPUT */  
    double *maxHeightFromSub /* OUTPUT */  
)
```

Parameters

layerId	Specifies the index into the database layer table.
minHeightFromSub	Specifies the minimum distance of the layer from the substrate.
nomHeightFromSub	Specifies the nominal distance of the layer from the substrate.
maxHeightFromSub	Specifies the maximum distance of the layer from the substrate.

Return

MWXSucces	Occurs if the data is successfully retrieved.
MWXNoLibraryOpened	Occurs if no library is open.
MWXFail	Occurs if the operation fails for any other reason.

Example

```
if (MWXdB_Get_LayerId_ByName ("metall", &layerId) == MWXSucces) {  
    if (MWXdB_Get_LayerHeightFromSub (layerId,  
        &minHeightFromSub, &nomHeightFromSub,  
        &maxHeightFromSub) == MWXSucces) {  
        rtn[0] = MWXSc_Make_DoubleSCM(minHeightFromSub);  
        rtn[1] = MWXSc_Make_DoubleSCM(nomHeightFromSub);  
        rtn[2] = MWXSc_Make_DoubleSCM(maxHeightFromSub);  
        return (MWXSc_Make_ListSCM (3, rtn));  
    }  
}
```

MWXdB_Get_LayerInductance

The MWXdB_Get_LayerInductance function gets the inductance values for a layer in a library. The associated keywords in the technology file are unitMinInductance, unitNomInductance, and unitMaxInductance.

Syntax

```
int MWXdB_Get_LayerInductance (
    int32 layerId, /* INPUT */
    double *minInductance, /* OUTPUT */
    double *nomInductance, /* OUTPUT */
    double *maxInductance /* OUTPUT */
)
```

Parameters

layerId	Specifies the index into the database layer table.
minInductance	Specifies the minimum inductance per unit length.
nomInductance	Specifies the nominal inductance per unit length.
maxInductance	Specifies the maximum inductance per unit length.

Return

MWXSucces	Occurs if the data is successfully retrieved.
MWXNoLibraryOpened	Occurs if no library is open.
MWXFail	Occurs if the operation fails for any other reason.

Example

```
if (MWXdB_Get_LayerId_ByName ("metall", &layerId) == MWXSucces) {
    if (MWXdB_Get_LayerInductance (layerId,
        &minInductance, &nomInductance,
        &maxInductance) == MWXSucces) {
        rtn[0] = MWXSc_Make_DoubleSCM(minInductance);
        rtn[1] = MWXSc_Make_DoubleSCM(nomInductance);
        rtn[2] = MWXSc_Make_DoubleSCM(maxInductance);
        return (MWXSc_Make_ListSCM (3, rtn));
    }
}
```

MWXdB_Get_LayerLayoutAttr

The `MWXdB_Get_LayerLayoutAttr` function gets the layout attributes for a layer in a library. The associated keywords in the technology file are

- `preferredDirection`
- `isDefaultLayer`
- `pitch`
- `minWidth`
- `defaultWidth`
- `minSpacing`
- `sameNetMinSpacing`

Syntax

```
int MWXdB_Get_LayerLayoutAttr {  
    int32 layerId, /* INPUT */  
    uchar *notUsed, /* OBSOLETE FIELD */  
    uchar *isDefaultLayer, /* OUTPUT */  
    double *pitch, /* OUTPUT */  
    double *minWidth, /* OUTPUT */  
    double *defaultWidth, /* OUTPUT */  
    double *spacing, /* OUTPUT */  
    double *sameNetSpacing /* OUTPUT */  
}
```

Parameters

<code>layerId</code>	Specifies the index into the database layer table.
<code>preferredDirection</code>	Not currently used.
<code>isDefaultLayer</code>	Specifies if the layer is the default routing layer when more than one layer has the same mask name (1 if true, 0 if false).
<code>pitch</code>	Specifies the distance between object centers.
<code>minWidth</code>	Specifies the minimum width of objects.
<code>defaultWidth</code>	Specifies the default width of objects.
<code>spacing</code>	Specifies the minimum distance between object edges.
<code>sameNetSpacing</code>	Specifies the minimum distance between edges of the same net.

Return

<code>MWXSucces</code>	Occurs if the data is successfully retrieved.
<code>MWXNoLibraryOpened</code>	Occurs if no library is open.
<code>MWXFail</code>	Occurs if the operation fails for any other reason.

Example

```
if (MWXDb_Get_LayerId_ByName ("metall", &layerId) == MWXSucces)
{
    if (MWXDb_Get_LayerLayoutAttr (layerId, &preferredDirection,
        &isDefaultLayer, &pitch, &minWidth, &defaultWidth,
        &spacing, &sameNetSpacing) == MWXSucces) {
        rtn[0] = MWXSc_Make_IntegerSCM(preferredDirection);
        rtn[1] = MWXSc_Make_BooleanSCM(isDefaultLayer);
        rtn[2] = MWXSc_Make_IntegerSCM(pitch);
        rtn[3] = MWXSc_Make_IntegerSCM(minWidth);
        rtn[4] = MWXSc_Make_IntegerSCM(defaultWidth);
        rtn[5] = MWXSc_Make_IntegerSCM(spacing);
        rtn[6] = MWXSc_Make_IntegerSCM(sameNetSpacing);
        return (MWXSc_Make_ListSCM (7, rtn));}
}
```

MWXDb_Get_LayerLineStyleName

The `MWXDb_Get_LayerLineStyleName` function gets the name of the layer line style for a layer in a library. The associated keyword in the technology file is `lineStyle`.

Syntax

```
int MWXDb_Get_LayerLineStyleName (  
    int32 layerId, /* INPUT */  
    char **lineStyleName /* OUTPUT */  
)
```

Parameters

layerId	Specifies the index into the database layer table.
lineStyleName	Specifies the name of the line style for the layer.

Return

MWXSuccess	Occurs if the data is successfully retrieved.
MWXNoLibraryOpened	Occurs if no library is open.
MWXFail	Occurs if the operation fails for any other reason.

Example

```
if (MWXDb_Get_LayerId_ByName ("metall", &layerId) == MWXSuccess) {  
    if (MWXDb_Get_LayerLineStyleName (layerId, &lineStyle) ==  
        MWXSuccess) {  
        rtn = MWXSc_Make_StringSCM (lineStyle);  
        free (lineStyle);  
        return (rtn);  
    }  
}
```

MWXDb_Get_LayerName_FromNumber

The `MWXDb_Get_LayerName_FromNumber` function gets the name of the layer identified by `layerNumber`. The library containing the layer information must be open for the operation to successfully execute.

Syntax

```
int MWXDb_Get_LayerName_FromNumber (  
    int32 layerNumber, /* INPUT */  
    char **layerName /* OUTPUT */  
)
```

Parameters

<code>layerNumber</code>	Specifies a unique number for the layer from the library technology data.
<code>layerName</code>	Specifies the name of the layer with the number <code>layerNumber</code> .

Return

<code>MWXSuccess</code>	Occurs if the layer name is successfully retrieved.
<code>MWXFail</code>	Occurs if the layer name retrieval fails.

Example

```
rc = MWXDb_Get_LayerName_FromNumber (layerNumber, &layerName);
    if (rc == MWXSuccess) {
        lname = MWXSc_Make_StringSCM (layerName);
        free (layerName);
        return (lname);
    } else {
        return (MWXFail);
    }
```

Scheme Equivalent

`dbGetLayerNameFromNumber` (see the *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Get_LayerNumber_FromName

The `MWXDb_Get_LayerNumber_FromName` function gets the number of the layer identified by `layerName`. The library containing the layer information must be open for the operation to successfully execute.

Syntax

```
int MWXDb_Get_LayerNumber_FromName (
    char *layerName,          /* INPUT */
    int32 *layerNumber        /* OUTPUT*/
)
```

Parameters

<code>layerName</code>	Specifies a unique name of the layer from the library technology data.
<code>layerNumber</code>	Specifies the number of the layer with the name <code>layerName</code> .

Return

<code>MWXSuccess</code>	Occurs if the layer number is successfully retrieved.
<code>MWXFail</code>	Occurs if the layer number retrieval fails.

Example

```
rc = MWXDb_Get_LayerNumber_FromName (layerName, &layerNumber);
    if (rc == MWXSuccess) {
        return (MWXSc_Make_IntegerSCM (layerNumber));
    } else {
        return (MWXFail);
    }
```

Scheme Equivalent

`dbGetLayerNumberFromName` (see the *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Get_LayerPhysicalAttr

The `MWXDb_Get_LayerPhysicalAttr` function gets the physical attributes for a layer in a library. The associated keywords in the technology file are

- `maxStackLevel`
- `maxSegLenForRC`
- `capacitanceMultiplier`
- `maxIntraCapDistRatio`
- `maxCurrDensity`

Syntax

```
int MWXDb_Get_LayerPhysicalAttr (
    int32 layerId, /* INPUT */
    int32 *maxStackLevel, /* OUTPUT */
    double *maxRCSegmentLength, /* OUTPUT
    used to be in INT32 type */
    int32 *capacitanceMultiplier, /* OUTPUT */
```

```

        int32 *maxIntraCapDistRatio, /* OUTPUT */
        double *maxCurrentDensity /* OUTPUT */
    )

```

Parameters

layerId	Specifies the index into the database layer table.
maxStackLevel	Specifies the maximum number of vias that can stack at the same point.
maxStackLevel	Specifies the maximum length of the wire segments on the layer.
maxStackLevel	Specifies the capacitance scaling factor.
maxStackLevel	Specifies the distance ratio required to control extraction.
maxStackLevel	Specifies the maximum current density of the layer.

Return

MWXSucces	Occurs if the data is successfully retrieved.
MWXNoLibraryOpened	Occurs if no library is open.
MWXFail	Occurs if the operation fails for any other reason.

Example

```

if (MWXDb_Get_LayerId_ByName ("metall", &layerId) == MWXSucces) {
    if (MWXDb_Get_LayerPhysicalAttr (layerId, &maxStackLevel,
        &maxRCSegmentLength, &capacitanceMultiplier,
        &maxIntraCapDistRatio, &maxCurrentDensity) ==
        MWXSucces) {
        rtn[0] = MWXSc_Make_IntegerSCM(maxStackLevel);
        rtn[1] = MWXSc_Make_IntegerSCM(maxRCSegmentLength);
        rtn[2] = MWXSc_Make_IntegerSCM(capacitanceMultiplier);
        rtn[3] = MWXSc_Make_IntegerSCM(maxIntraCapDistRatio);
        rtn[4] = MWXSc_Make_DoubleSCM(maxCurrentDensity);
        return (MWXSc_Make_ListSCM (5, rtn));
    }
}

```

MWXDb_Get_LayerResistance

The `MWXDb_Get_LayerResistance` function gets the resistance values for a layer in a library. The associated keywords in the technology file are `unitMinResistance`, `unitNomResistance`, and `unitMaxResistance`.

Syntax

```
int MWXDb_Get_LayerResistance (
    int32 layerId, /* INPUT */
    double *minResistance, /* OUTPUT */
    double *nomResistance, /* OUTPUT */
    double *maxResistance /* OUTPUT */
)
```

Parameters

layerId	Specifies the index into the database layer table.
minResistance	Specifies the minimum resistance per square.
nomResistance	Specifies the nominal resistance per square.
maxResistance	Specifies the maximum resistance per square.

Return

MWXSuccess	Occurs if the data is successfully retrieved.
MWXNoLibraryOpened	Occurs if no library is open.
MWXFail	Occurs if the operation fails for any other reason.

Example

```
if (MWXDb_Get_LayerId_ByName ("metall", &layerId) == MWXSuccess) {
    if (MWXDb_Get_LayerResistance (layerId, &minResistance,
        &nomResistance, &maxResistance) == MWXSuccess) {
        rtn[0] = MWXSc_Make_DoubleSCM(minResistance);
        rtn[1] = MWXSc_Make_DoubleSCM(nomResistance);
        rtn[2] = MWXSc_Make_DoubleSCM(maxResistance);
        return (MWXSc_Make_ListSCM (3, rtn));}
}
```

MWXDb_Get_LayerStippleName

The MWXDb_Get_LayerStippleName function gets the name of the layer stipple pattern for a layer in a library. The associated keyword in the technology file is pattern.

Note:

There is no need to free stippleName.

Syntax

```
int MWXDb_Get_LayerStippleName (
    int32 layerId, /* INPUT */
    char **stippleName /* OUTPUT */
)
```

Parameters

<code>layerId</code>	Specifies the index into the database layer table.
<code>stippleName</code>	Specifies the name of the stipple pattern for the layer.

Return

<code>MWXSucces</code>	Occurs if the data is successfully retrieved.
<code>MWXNoLibraryOpened</code>	Occurs if no library is open.
<code>MWXFail</code>	Occurs if the operation fails for any other reason.

Example

```
if (MWXDb_Get_LayerId_ByName ("metall", &layerId) == MWXSucces) {
    if (MWXDb_Get_LayerStippleName (layerId, &stippleName) ==
        MWXSucces) {
        rtn = MWXSc_Make_StringSCM (stippleName);
        return (rtn);
    }
}
```

MWXDb_Get_LayerStubRouting

The `MWXDb_Get_LayerStubRouting` function gets the stub routing factors for a layer in a library. The associated keywords in the technology file are `stubThreshold`, `stubSpacing`, and `maxLength`.

Syntax

```
int MWXDb_Get_LayerStubRouting (
    int32 layerId, /* INPUT */
    double *stubThreshold, /* OUTPUT */====\
    double *stubSpacing, /* OUTPUT */====\
        used to be in INT32 type
    double *maxLength /* OUTPUT */====/
)
```


Parameters

<code>layerId</code>	Specifies the index into the database layer table.
<code>stubThreshold</code>	Specifies the distance for which two objects must remain parallel before the <code>stubSpacing</code> specification applies.
<code>stubSpacing</code>	Specifies the minimum distance between two parallel objects that remain parallel for a distance equal to or greater than <code>stubThreshold</code> .
<code>maxLength</code>	Specifies the maximum length of an object on the layer.

Return

<code>MWXSuccess</code>	Occurs if the data is successfully retrieved.
<code>MWXNoLibraryOpened</code>	Occurs if no library is open.
<code>MWXFail</code>	Occurs if the operation fails for any other reason.

Example

```
if (MWXDb_Get_LayerId_ByName ("metall", &layerId) == MWXSuccess) {
    if (MWXDb_Get_LayerStubRouting (layerId, &stubThreshold,
        &stubSpacing, &maxLength) == MWXSuccess) {
        rtn[0] = MWXSc_Make_IntegerSCM(stubThreshold);
        rtn[1] = MWXSc_Make_IntegerSCM(stubSpacing);
        rtn[2] = MWXSc_Make_IntegerSCM(maxLength);
        return (MWXSc_Make_ListSCM (3, rtn));
    }
}
```

MWXDb_Get_LayerThickness

The `MWXDb_Get_LayerThickness` function gets the thickness values for a layer in a library. The associated keywords in the technology file are `unitMinThickness`, `unitNomThickness`, and `unitMaxThickness`.

Syntax

```
int MWXDb_Get_LayerThickness (
    int32 layerId, /* INPUT */
    double *minThickness, /* OUTPUT */
    double *nomThickness, /* OUTPUT */
    double *maxThickness /* OUTPUT */
)
```

Parameters

<code>layerId</code>	Specifies the index into the database layer table.
<code>minThickness</code>	Specifies the minimum thickness for objects on the layer.
<code>nomThickness</code>	Specifies the nominal thickness for objects on the layer.
<code>maxThickness</code>	Specifies the maximum thickness for objects on the layer.

Return

<code>MWXSuccess</code>	Occurs if the data is successfully retrieved.
<code>MWXNoLibraryOpened</code>	Occurs if no library is open.
<code>MWXFail</code>	Occurs if the operation fails for any other reason.

Example

```
if (MWXDb_Get_LayerId_ByName ("metall", &layerId) == MWXSuccess) {
    if (MWXDb_Get_LayerThickness (layerId, &minThickness,
        &nomThickness, &maxThickness) == MWXSuccess) {
        rtn[0] = MWXSc_Make_DoubleSCM(minThickness);
        rtn[1] = MWXSc_Make_DoubleSCM(nomThickness);
        rtn[2] = MWXSc_Make_DoubleSCM(maxThickness);
        return (MWXSc_Make_ListSCM (3, rtn));
    }
}
```

MWXDb_Get_TechCapacitance

The `MWXDb_Get_TechCapacitance` function retrieves the `capacitancePrecision` and `unitCapacitanceName` values from the technology file.

Syntax

```
int MWXDb_Get_TechCapacitance (
    int32  *capacitancePrecision /* OUTPUT */
    char   **unitCapacitanceName /* OUTPUT */
)
```

Parameters

capacitancePrecision	Defines how accurately capacitance values are stored in the database.
unitCapacitanceName	The unit of capacitance. Valid values are ff, pf, nf, uf, mf, and f.

Return

MWXNoLicense	Occurs if the Synopsys license is not present.
MWXNoLibraryOpened	Occurs if no library is open. No data is retrieved.
MWXSuccess	Occurs if the data is successfully retrieved.

Example

```
rc = MWXDb_Get_TechCapacitance(&capacitancePrecision,
&unitCapacitanceName);
    switch (rc) {
        case MWXSuccess:
            rtn[0]= MWXSc_Make_IntegerSCM
                (capacitancePrecision);
            rtn[1]= MWXSc_Make_StringSCM
                (unitCapacitanceName);
            return (MWXSc_Make_ListSCM (2, rtn));
        default:
            MWX_Printf("ERROR -- could not retrieve
                data\n");
            return (MWXFail);
    }
```

MWXDb_Get_TechContactLayer

The MWXDb_Get_TechContactLayer function gets a contact layer such as viaLayerNumber, lowerLayerNumber, or upperLayerNumber from the technology file.

Syntax

```
int MWXDb_Get_TechContactLayer (
    uchar contactCodeIdx, /* INPUT */
    int32 *viaLayerNumber, /* OUTPUT */
    int32 *lowerLayerNumber, /* OUTPUT */
    int32 *upperLayerNumber /* OUTPUT */
)
```

Parameters

contactCodeIdx	Specifies the index of the contact code for which the contact size is retrieved.
viaLayerNumber	The number of viaLayer.
lowerLayerNumber	The number of lowerLayer.
upperLayerNumber	The number of upperLayer.

Return

MWXSucces	Occurs if the contact layer is successfully retrieved.
MWXFail	Occurs if the contact layer is not successfully retrieved.

Example

```
if (MWXDb_Get_TechContactLayer(contactCodeId, &viaLayerNumber,
&lowerLayerNumber, &upperLayerNumber) != MWXSucces) {
    MWX_Printf("Fail to get TechContactLayer \n");
    return (MWXFail);
}
rtn[0] = MWXSc_Make_IntegerSCM(viaLayerNumber);
rtn[1] = MWXSc_Make_IntegerSCM(lowerLayerNumber);
rtn[2] = MWXSc_Make_IntegerSCM(upperLayerNumber);
return (MWXSc_Make_ListSCM (3, rtn));
```

MWXDb_Get_TechContactSize

The MWXDb_Get_TechContactSize function gets a contact size such as viaSizeX, viaSizeY, lowerEnclosureX, lowerEnclosureY, upperEnclosureX, upperEnclosureY, or viaSpacing from the technology file.

Syntax

```
int MWXDb_Get_TechContactSize (
    uchar contactCodeIdx, /* INPUT */
    int32 *viaSizeX, /* OUTPUT */
    int32 *viaSizeY, /* OUTPUT */
    int32 *lowerEnclosureX, /* OUTPUT */
    int32 *lowerEnclosureY, /* OUTPUT */
    int32 *upperEnclosureX, /* OUTPUT */
    int32 *upperEnclosureY, /* OUTPUT */
    int32 *viaSpacing /* OUTPUT */
)
```

Parameters

contactCodeIdx	Specifies the index of the contact code for which the contact size are retrieved.
viaSizeX	Size of the via on the x-coordinate.
viaSizeY	Size of the via on the y-coordinate.
lowerEnclosureX	Size of the lower enclosure on the x-coordinate.
lowerEnclosureY	Size of the lower enclosure on the y-coordinate.
upperEnclosureX	Size of the upper enclosure on the x-coordinate.
upperEnclosureY	Size of the upper enclosure on the y-coordinate.
viaSpacing	Size of the via spacing.

Return

MWXSucces	Occurs if the contact size is successfully retrieved.
MWXFail	Occurs if the contact size is not successfully retrieved.

Example

```
if (MWXDb_Get_TechContactSize(contactCodeId, &viaSizeX, &viaSizeY,
    &lowerEnclosureX, &lowerEnclosureY, &upperEnclosureX, &upperEnclosureY,
    &viaSpacing) != MWXSucces) {
MWX_Printf("Fail to get TechContactSize bit \n");
return (MWXFail);
}
    rtn[0] = MWXSc_Make_IntegerSCM(viaSizeX);
rtn[1] = MWXSc_Make_IntegerSCM(viaSizeY);
rtn[2] = MWXSc_Make_IntegerSCM(lowerEnclosureX);
rtn[3] = MWXSc_Make_IntegerSCM(lowerEnclosureY);
rtn[4] = MWXSc_Make_IntegerSCM(upperEnclosureX);
rtn[5] = MWXSc_Make_IntegerSCM(upperEnclosureY);
rtn[6] = MWXSc_Make_IntegerSCM(viaSpacing);
return (MWXSc_Make_ListSCM (7, rtn));
```

MWXDb_Get_TechCurrent

The MWXDb_Get_TechCurrent function retrieves the CurrentPrecision and CurrentUnit values from the technology file.

Syntax

```
#include "mwx_proto.h"
int MWXDb_Get_TechCurrent (
    int32 *CurrentPrecision /* Output */
    char **CurrentUnit /* Output */
)
```

Parameters

CurrentPrecision	Defines how accurately current measurements are stored in the database.
CurrentUnit	The unit of current. Valid values are fA,pA, nA, uA, mA, and A.

Return

MWXNoLibraryOpened	Occurs if no library is open. No data is retrieved.
MWXSuccess	Occurs if database data is retrieved.
MWXNoLicense	Occurs if the Synopsys license is not present.

Dynamic Example

```
switch (rc) {
    case MWXSuccess:
        rtn[0]= MWXSc_Make_IntegerSCM
            (CurrentPrecision);
        rtn[1]= MWXSc_Make_StringSCM (CurrentUnit);
        return (MWXSc_Make_ListSCM (2, rtn));
    case MWXNoLibraryOpened:
        MWX_Printf("ERROR -- library not open\n");
        return (MWXFail);
}
```

MWXDb_Get_TechDielectric

The MWXDb_Get_TechDielectric function retrieves the dielectric constant from the technology file.

Syntax

```
int MWXDb_Get_TechDielectric (
    double *dielectric /* OUTPUT */
)
```

dielectric SiO2 dielectric constant specified in the technology file.

MWXNoLicense	Occurs if the Synopsys license is not present.
MWXNoLibraryOpened	Occurs if no library is open. No data is retrieved.
MWXSuccess	Occurs if the data is successfully retrieved.

```
rc = MWXDb_Get_TechDielectric (&dielectric);
    switch (rc) {
        case MWXSuccess:
            return (MWXSc_Make_DoubleSCM (dielectric));
        default:
            MWX_Printf("ERROR -- could not retrieve
            data\n");
            return (MWXFail);
    }
}
```

The `MWXDb_Get_TechInductance` function gets the `InductancePrecision` and `InductanceUnit` values from the technology file.

```
#include "mw_x_proto.h"
int MWXDb_Get_TechInductance (
    int32_t *InductancePrecision /* Output */
    char **InductanceUnit /* Output */
)
```

Parameters

InductancePrecision	Defines how accurately inductance measurements are stored in the database.
InductanceUnit	The unit of inductance. Valid values are fh, ph, nh, uh, mh, and h.

Return

MWXNoLibraryOpened	Occurs if no library is open. No data is retrieved.
MWXSuccess	Occurs if data is successfully retrieved from the database.
MWXNoLicense	Occurs if the Synopsys license is not present.

Dynamic Example

```
switch (rc) {
    case MWXSuccess:
        rtn[0]= MWXSc_Make_IntegerSCM
            (InductancePrecision);
        rtn[1]= MWXSc_Make_StringSCM (InductanceUnit);
        return (MWXSc_Make_ListSCM (2, rtn));
    case MWXNoLibraryOpened:
        MWX_Printf("ERROR -- library not open\n");
        return (MWXFail);
}
```

MWXDb_Get_TechName

The MWXDb_Get_TechName function gets the TechName value from the technology file.

Note:

The string produced by this function needs to be copied over to another temporary string before you perform operations such as strtok on the string. Otherwise, after MWXDb_Get_TechName() is invoked once, it creates an invalid string if invoked again during the same Milkyway/Astro session.

Syntax

```
int32 MWXDb_Get_TechName (
    char    **name /* Output */
)
```


Parameters

name Name of the technology file.

Return

MWXSuccess Occurs if data is successfully retrieved from the database.

MWXFail Occurs if the operation fails for any other reason.

Dynamic Example

```
MWXSCM_t mwTechName (MWXSCM_t list)
{
    int rc;
    char *name;
    MWX_Printf("INFO -- getting the TechName ...\n");
    rc = MWXDb_Get_TechName(&name);
    switch (rc)
    {
        case MWXSuccess:
        {
            return (MWXSc_Make_StringSCM(name));
        }
        case MWXNoLibraryOpened:
            MWX_Printf("ERROR -- library not open\n");
            return (MWXFail);
        default:
            MWX_Printf("ERROR -- unknown return code from
            MWXDb_Get_TechName\n");
            return (MWXFail);
    }
}
```

MWXDb_Get_TechPower

The MWXDb_Get_TechPower function retrieves the powerPrecision and unitPowerName values from the technology file.

Syntax

```
int MWXDb_Get_TechPower (
    int32 powerPrecision /* OUTPUT */
    char unitPowerName /* OUTPUT */
)
```

Parameters

<code>powerPrecision</code>	Defines how accurately power values are stored in the database.
<code>unitPowerName</code>	Defines the unit of power. Valid values are <code>fw</code> , <code>pw</code> , <code>nw</code> , <code>uw</code> , <code>mw</code> , and <code>w</code> .

Return

<code>MWXNoLicense</code>	Occurs if the Synopsys license is not present.
<code>MWXNoLibraryOpened</code>	Occurs if no library is open. No data is retrieved.
<code>MWXSuccess</code>	Occurs if the data is successfully retrieved.

MWXDb_Get_TechResistance

The `MWXDb_Get_TechResistance` function gets the `ResistancePrecision` and `ResistanceUnit` values from the technology file.

Syntax

```
#include "mwx_proto.h"
int MWXDb_Get_TechResistance (
    int32 *ResistancePrecision /* Output */
    char **ResistanceUnit /* Output */
)
```

Parameters

ResistancePrecision	Defines how accurately resistance measurements are stored in the database.
ResistanceUnit	The unit of resistance. Valid values are mohm, ohm, kohm, and Mohm.

Return

MWXNoLicense	Occurs if the Synopsys license is not present.
MWXNoLibraryOpened	Occurs if no library is open. No data is retrieved.
MWXSuccess	Occurs if the data is successfully retrieved from the database.

Dynamic Example

```
switch (rc) {
    case MWXSuccess:
        rtn[0]= MWXSc_Make_IntegerSCM
            (ResistancePrecision);
        rtn[1]= MWXSc_Make_StringSCM (ResistanceUnit);
        return (MWXSc_Make_ListSCM (2, rtn));
    case MWXNoLibraryOpened:
        MWX_Printf("ERROR -- library not open\n");
        return (MWXFail);
}
```

MWXDb_Get_TechResolution

The MWXDb_Get_TechResolution function gets the lengthPrecision, minimumResolution and lengthUnit values from database.

Syntax

```
#include "mwx_proto.h"
int MWXDb_Get_TechResolution (
    int32      *lengthPrecision /*Output*/
    int32      *minimumResolution /*Output*/
    char       **lengthUnit /*Output*/
)
```

Parameters

<code>lengthPrecision</code>	The length of the grid.
<code>minimumResolution</code>	The minimum grid resolution.
<code>lengthUnit</code>	The unit of length. The valid unit is the micron.

Return

<code>MWXNoLicense</code>	Occurs if the Synopsys license is not present.
<code>MWXNoLibraryOpened</code>	Occurs if no library is open. No data is retrieved.
<code>MWXSuccess</code>	Occurs if the data is successfully retrieved from the database.

Dynamic Example

```
rc = MWXDb_Get_TechResolution(&lengthPrecision,  
                             &minimumResolution, &lengthUnit);  
switch (rc) {  
    case MWXSuccess:  
    {  
        rtn[0] = MWXSc_Make_IntegerSCM  
            (lengthPrecision);  
        rtn[1] = MWXSc_Make_IntegerSCM  
            (minimumResolution);  
        rtn[2] = MWXSc_Make_StringSCM(lengthUnit);  
        return (MWXSc_Make_ListSCM (3, rtn) );  
    }  
    case MWXNoLibraryOpened:  
        MWX_Printf("ERROR -- library not open\n");  
        return (MWXFail);  
}
```

MWXDb_Get_TechRowAbutable

The `MWXDb_Get_TechRowAbutable` function gets the row setting information.

Syntax

```
int MWXDb_Get_TechRowAbutable (  
    uchar *tbRowAbutable,  
    uchar *tbRowAbutable,  
    uchar *bbRowAbutable,  
)
```

Parameters

ttRowAbuttable	Indicates whether top and top edges can be abutted when you are using double-back cell rows. Valid values: 1 (yes) and 0 (no).
tbRowAbuttable	Indicates whether top and bottom edges can be abutted when you are using double-back cell rows. Valid values: 1 (yes) and 0 (no).
bbRowAbuttable	Indicates whether bottom and bottom edges can be abutted when you are using double-back cell rows. Valid values: 1 (yes) and 0 (no).

Return

MWXNoLibraryOpened	Occurs if the library is not open.
MWXSuccess	Occurs if the RowAbutt information is successfully retrieved from the technology file.

Example

```
rc = MWXDb_Get_TechRowSpacing ( &ttRowSpacing, &bbRowSpacing, &tbRowSpacing);
    switch (rc) {
        case MWXSuccess:
            return (MWXSuccess)
        case MWXFail:
            MWX_Printf("ERROR -- Failed to operate. \n");
            return (MWXFail);
    }
```

MWXDb_Get_TechRowSpacing

The MWXDb_Get_TechRowSpacing function gets the row spacing setting information.

Syntax

```
int MWXDb_Get_TechRowSpacing (
    int32 *ttRowSpacing,
    int32 *tbRowSpacing,
    int32 *bbRowSpacing,
)
```

Parameters

<code>ttRowSpacing</code>	Defines the spacing between the top edges when you are using double-back cell rows in the floor plan.
<code>tbRowSpacing</code>	Defines the spacing between the top and bottom edges when you are using double-back cell rows in the floor plan.
<code>bbRowSpacing</code>	Defines the spacing between the bottom edges when you are using double-back cell rows in the floor plan.

Return

<code>MWXNoLibraryOpened</code>	Occurs if the library is not open.
<code>MWXSuccess</code>	Occurs if the spacing setting information is successfully retrieved from technology file.

Example

```
rc = MWXDb_Get_TechRowSpacing ( &ttRowSpacing, &tbRowSpacing, &bbRowSpacing );
    switch (rc) {
        case MWXSuccess:
            return (MWXSuccess)
        case MWXFail:
            MWX_Printf("ERROR -- Failed to operate. \n");
            return (MWXFail);
    }
```

MWXDb_Get_TechTime

The `MWXDb_Get_TechTime` function retrieves the `timePrecision` and `unitTimeName` values from the technology file.

Syntax

```
int MWXDb_Get_TechTime (
    char    **unitTimeName /* OUTPUT */
)
```

Parameters

<code>timePrecision</code>	Defines how accurately time values are stored in the database.
<code>unitTimeName</code>	The unit of time. Valid values are <code>fs</code> , <code>ps</code> , <code>ns</code> , <code>us</code> , <code>ms</code> , and <code>s</code> .

Return

<code>MWXNoLicense</code>	Occurs if the Synopsys license is not present.
<code>MWXNoLibraryOpened</code>	Occurs if no library is open. No data is retrieved.
<code>MWXSuccess</code>	Occurs if the data is successfully retrieved.

Dynamic Example

```
rc = MWXDb_Get_TechTime( &timePrecision, &unitTimeName);
    switch (rc) {
        case MWXSuccess:
            rtn[0]= MWXSc_Make_IntegerSCM (timePrecision);
            rtn[1]= MWXSc_Make_StringSCM (unitTimeName);
            return (MWXSc_Make_ListSCM (2, rtn));
        default:
            MWX_Printf("ERROR -- could not retrieve
            data\n");
            return (MWXFail);
    }
```

MWXDb_Get_TechVoltage

The `MWXDb_Get_TechVoltage` function retrieves the `voltagePrecision` and `unitVoltageName` values from the technology file.

Syntax

```
int MWXDb_Get_TechVoltage (
    int32  *voltagePrecision /* OUTPUT */
    char   **unitVoltageName /* OUTPUT */
)
```

Parameters

<code>voltagePrecision</code>	Defines how accurately voltage values are stored in the database.
<code>unitVoltageName</code>	The unit of voltage. Valid values are <code>fV</code> , <code>pV</code> , <code>nV</code> , <code>uV</code> , <code>mV</code> , and <code>V</code> .

Return

<code>MWXNoLicense</code>	Occurs if the Synopsys license is not present.
<code>MWXNoLibraryOpened</code>	Occurs if no library is open. No data is retrieved.
<code>MWXSuccess</code>	Occurs if the data is successfully retrieved.

Dynamic Example

```
rc = MWXDb_Get_TechVoltage( &voltagePrecision,
                           &unitVoltageName );
switch (rc) {
    case MWXSuccess:
        rtn[0]= MWXSc_Make_IntegerSCM
            (voltagePrecision);
        rtn[1]= MWXSc_Make_StringSCM
            (unitVoltageName);
        return (MWXSc_Make_ListSCM (2, rtn));
    default:
        MWX_Printf("ERROR -- could not retrieve
            data\n");
        return (MWXFail);
}
```


6

Scheme I/O and Boolean Functions

This chapter describes the Scheme I/O and the Boolean functions.

MWX_Log

Writes a formatted text string to the log file. This function provides the formatting characteristics of the C `printf` function.

Syntax

```
#include "mwx.h"
void MWX_Log(char *ptr, ...)
```

Parameters

`ptr` Specifies a string to write to the log file.

Return

None used

Example

```
MWX_Log("Cell name: '%s'; number of ports: '%d'\n", cellName, portCount);
```

MWX_Printf

Writes a formatted text string to the tool log window and log file. This function provides the formatting characteristics of the C `printf` function.

Syntax

```
#include "mwx.h"
void MWX_Printf(char *ptr, ...)
```

Parameters

`ptr` Specifies a string to print.

Return

None used

Example

```
MWX_Printf("Cell name: '%s'; number of ports: '%d'\n", cellName, portCount);
```

MWXDb_BooleanAnd_Objects

Performs the Boolean AND operation on the two input lists of polygons (`polygon1` and `polygon2`) that are contained in the cell identified as `cellId` to produce the output list of polygons called `newPoly`. Each of the input lists of polygons should end with a 0. It is your responsibility to free the memory of the return object's array.

Syntax

```
int MWXDb_BooleanAnd_Objects (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t  *polygon1, /* INPUT */
    MWXObjId_t  *polygon2, /* INPUT */
    MWXObjId_t  **newPoly /*OUTPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell containing objects to Boolean AND.
<code>polygon1</code>	Specifies the database identifier of the first input list of polygons.
<code>polygon1</code>	Specifies the database identifier of the second input list of polygons.
<code>newPoly</code>	Specifies the database identifier of the output list of resulting Boolean AND polygons.

Return

<code>MWXSuccess</code>	Occurs if the two input lists of polygons are successfully converted to Boolean AND.
<code>MWXFail</code>	Occurs if the two input lists of polygons are not successfully converted to Boolean AND.

Example

```
rc = MWXDb_BooleanAnd_Objects(cellId, polyId1, polyId2, &newPolyId);
switch (rc) {
    case MWXSuccess:
        MWX_Printf("SUCCESS...The New Polygon Id is:
            %d \n", *newPolyId);
        return (MWX_TRUE_SCM);
    case MWXFail:
        MWX_Printf("FAILURE...Fail to booleanAnd the
            two provided lists of objects. \n");
        return (MWX_FALSE_SCM);
    default:
        return (MWX_FALSE_SCM);
}
```

```
}
```

MWXDb_BooleanNot_Objects

Performs the Boolean NOT operation on the two input lists of polygons (`polygon1` and `polygon2`) that are contained in the cell identified as `cellId` to produce the output list of polygons called `newPoly`. Each of the input lists of polygons should end with a 0. It is your responsibility to free the memory of the return object's array.

Syntax

```
int MWXDb_BooleanNot_Objects (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t *polygon1, /* INPUT */
    MWXObjId_t *polygon2, /* INPUT */
    MWXObjId_t **newPoly /*OUTPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell containing objects to Boolean NOT.
<code>polygon1</code>	Specifies the database identifier of the first input list of polygons.
<code>polygon2</code>	Specifies the database identifier of the second input list of polygons.
<code>newPoly</code>	Specifies the database identifier of the output list of resulting Boolean NOT polygons.

Return

<code>MWXSuccess</code>	Occurs if the two input lists of polygons are successfully converted to Boolean NOT.
<code>MWXFail</code>	Occurs if the two input lists of polygons are not successfully converted to Boolean NOT.

Example

```
rc = MWXDb_BooleanNot_Objects(cellId, polyId1, polyId2, &newPolyId);
switch (rc) {
    case MWXSuccess:
        MWX_Printf("SUCCESS...The New Polygon Id is:
            %d \n", *newPolyId);
        return (MWX_TRUE_SCM);
    case MWXFail:
        MWX_Printf("FAILURE...Fail to booleanNot the
            two provided lists of objects. \n");
        return (MWX_FALSE_SCM);
}
```

```

default:
    return (MWX_FALSE_SCM);

```

MWXDb_BooleanOr_Objects

Performs the Boolean OR operation on the two input lists of polygons (`polygon1` and `polygon2`) that are contained in the cell identified as `cellId` to produce the output list of polygons called `newPoly`. Each of the input lists of polygons should end with a 0. It is your responsibility to free the memory of the return object's array.

Syntax

```

int MWXDb_BooleanOr_Objects (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t *polygon1, /* INPUT */
    MWXObjId_t *polygon2, /* INPUT */
    MWXObjId_t **newPoly /*OUTPUT */
)

```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell containing objects to Boolean OR.
<code>polygon1</code>	Specifies the database identifier of the first input list of polygons.
<code>polygon2</code>	Specifies the database identifier of the second input list of polygons.
<code>newPoly</code>	Specifies the database identifier of the output list of resulting Boolean OR polygons.

Return

<code>MWXSuccess</code>	Occurs if the Boolean OR operation is successful.
<code>MWXFail</code>	Occurs if the Boolean OR operation is not successful.

Example

```

rc = MWXDb_BooleanOr_Objects(cellId, polyId1, polyId2, &newPolyId);
switch (rc) {
    case MWXSuccess:
        MWX_Printf("SUCCESS...The New Polygon Id is:
            %d \n", *newPolyId);
        return (MWX_TRUE_SCM);
    case MWXFail:
        MWX_Printf("FAILURE...Fail to booleanOr the
            two provided lists of objects. \n");
        return (MWX_FALSE_SCM);
    default:

```

```

        return (MWX_FALSE_SCM);
    }

```

MWXDb_BooleanXor_Objects

Performs the Boolean XOR operation on the two input lists of polygons (*polygon1* and *polygon2*) that are contained in the cell identified as *cellId* to produce the output list of polygons called *newPoly*. Each of the input lists of polygons should end with a 0. It is your responsibility to free the memory of the return object's array.

Syntax

```

int MWXDb_BooleanXor_Objects (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t *polygon1, /* INPUT */
    MWXObjId_t *polygon2, /* INPUT */
    MWXObjId_t **newPoly, /*OUTPUT */
)

```

Parameters

<i>cellId</i>	Specifies the database identifier of the cell containing objects to Boolean XOR.
<i>polygon1</i>	Specifies the database identifier of the first input list of polygons.
<i>polygon1</i>	Specifies the database identifier of the second input list of polygons.
<i>newPoly</i>	Specifies the database identifier of the output list of resulting Boolean XOR polygons.

Return

<i>MWXSuccess</i>	Occurs if the two input lists of polygons are successfully converted to Boolean XOR.
<i>MWXFail</i>	Occurs if the two input lists of polygons are not successfully converted to Boolean XOR.

Example

```

rc = MWXDb_BooleanXor_Objects(cellId, polyId1, polyId2, &newPolyId);
switch (rc) {
    case MWXSuccess:
        MWX_Printf("SUCCESS...The New Polygon Id is:
            %d \n", *newPolyId);
        return (MWX_TRUE_SCM);
    case MWXFail:
        MWX_Printf("FAILURE...Fail to booleanXor the
            two provided lists of objects. \n");

```

```
        return (MWX_FALSE_SCM);  
default:  
    return (MWX_FALSE_SCM);  
}
```


7

Scheme Interpreter Functions

This chapter describes the `MWXS_C_*` functions, which relate to direct communication between C modules and the Scheme interpreter.

Classification of Scheme Interpreter Functions

The Scheme interpreter functions are classified as follows:

- Miscellaneous `MWXSc_*` procedures, used to get values, send commands to the Scheme interpreter, assign names to procedures, and so on:
 - [MWXSc_Car](#)
 - [MWXSc_Cdr](#)
 - [MWXSc_Evaluate](#)
 - [MWXSc_Register_Function](#)
 - [MWXSc_Scanf](#)

The first two functions operate on Scheme lists (`MWXSCM_t`) in the C environment, much like the Scheme commands `car` and `cdr`.

- `MWXSc_Get_*` procedures, used to translate data coming from Scheme to the C data types, which include
 - [MWXSc_Get_BooleanSCM](#)
 - [MWXSc_Get_DBIdSCM](#)
 - [MWXSc_Get_DoubleSCM](#)
 - [MWXSc_Get_IntegerSCM](#)
 - [MWXSc_Get_StringSCM](#)
 - [MWXSc_Get_Symbol](#)
- `MWXSc_Is_*` procedures, used to test if a Scheme data item is of a particular type, which include
 - [MWXSc_Is_BooleanSCM](#)
 - [MWXSc_Is_DoubleSCM](#)
 - [MWXSc_Is_DBIdSCM](#)
 - [MWXSc_Is_IntegerSCM](#)
 - [MWXSc_Is_ListSCM](#)
 - [MWXSc_Is_NullSCM](#)
 - [MWXSc_Is_StringSCM](#)

- `MWXSc_Make_*` procedures, used to translate the data sent from C to Scheme, which include
 - [MWXSc_Make_BooleanSCM](#)
 - [MWXSc_Make_DBIdSCM](#)
 - [MWXSc_Make_DoubleSCM](#)
 - [MWXSc_Make_IntegerSCM](#)
 - [MWXSc_Make_ListSCM](#)
 - [MWXSc_Make_StringSCM](#)

Scheme Function Descriptions

This section describes the Scheme interpreter functions.

MWXSc_Car

Gets the Scheme value at the head of a list.

Syntax

```
int32 MWXSc_Car (  
    MWXSCM_t scm  
)
```

Parameters

<code>scm</code>	Specifies the Scheme list.

Return

If the list is not empty, the Scheme value at the head of the list will be returned. If the list is empty, an empty list will be returned.

Example

```
if (MWXSc_Is_ListSCM(scmVal) == MWXSuccess) {  
    for (l = scmVal; ! MWXSc_Is_NullSCM(l); l = MWXSc_Cdr(l)) {  
        scmVal = MWXSc_Car(l);  
        .  
        .  
        .  
    }  
}
```

Scheme Equivalent

`car` (see the *Scheme Language Reference Manual*)

MWXSc_Cdr

Gets the list of values following the first value of a list.

Syntax

```
int32 MWXSc_Cdr (  
    MWXSCM_t scm  
)
```

Parameters

`scm` Specifies the Scheme list.

Return

If the list has more than one element, a list of all elements following the first is returned. If there are no elements or only one, an empty list is returned.

Example

```
if (MWXSc_Is_ListSCM(scmVal) == MWXSuccess) {  
    for (l = scmVal; ! MWXSc_Is_NullSCM(l); l = MWXSc_Cdr(l)) {  
        scmVal = MWXSc_Car(l);  
        .  
        .  
        .  
    }  
}
```

Scheme Equivalent

`cdr` (see the *Scheme Language Reference Manual*)

MWXSc_Evaluate

Sends a command to the Scheme interpreter. The return value is not the result of the Scheme command. Preserve the result of issuing the Scheme command by assigning it to a variable. You can retrieve the variable value later by using the `MWXSc_Get_Symbol` function.

Syntax

```
int32 MWXSc_Evaluate(  
    char *scm_cmd  
)
```

Parameters

scm_cmd Specifies the Scheme command to execute.

Return

MWXSuccess Occurs if the command is successfully passed to the Scheme interpreter.

MWXNoLicense Occurs if no license is available for execution.

Example

```
rc = MWXSc_Evaluate("(define cell (dbOpenCell \"mycell\" \"r\"));  
if (rc != MWXSuccess) {  
    /* Evaluate failed */  
    .  
    .  
    .  
}  
else {  
    symbol = MWXSc_Get_Symbol("cell");  
    /* If the result is boolean, it must be false and the operation failed. */  
    if (MWXSc_Is_BooleanSCM(symbol)) {  
        .  
        .  
        .  
    }  
    /* If the result is a database identifier, the operation succeeded. */  
    else if (MWXSc_Is_DBIIdSCM(symbol)) {  
        MWXSc_Get_DBIIdSCM(symbol, &DBId);  
        .  
        .  
        .  
    }  
    /* Any other data type and the operation failed. */  
    else {  
        .  
        .  
        .  
    }  
}
```

MWXSc_Get_BooleanSCM

Translates the Scheme Boolean value into an equivalent C value (0 for false, nonzero for true).

Syntax

```
int32 MWXSc_Get_BooleanSCM (
    MWXSCM_t scm,
    unsigned char *pBoolean;
)
```

Parameters

<code>scm</code>	Specifies the Scheme Boolean value.
<code>pBoolean</code>	Specifies the address in which to store the C Boolean equivalent value of <code>scm</code> .

Return

<code>MWXSuccess</code>	Occurs if the translation is successful.
<code>MWXWrongSymbolType</code>	Occurs if the value is not Boolean.

Example

```
if (MWXSc_Is_BooleanSCM(scmVal) == MWXSuccess) {
    MWXSc_Get_BooleanSCM(scmVal, &cVal);
}
```

MWXSc_Get_DBIdSCM

Translates the Scheme database identifier into an equivalent C value.

Syntax

```
int32 MWXSc_Get_DBIdSCM (
    MWXSCM_t scm,
    MWXDBId_t *pDBId
)
```

Parameters

<code>scm</code>	Specifies the Scheme database identifier.
<code>pDBId</code>	Specifies the address in which to store the C database identifier equivalent value of <code>scm</code> .

Return

<code>MWXSuccess</code>	Occurs if the translation is successful.
<code>MWXWrongSymbolType</code>	Occurs if the value is not a database identifier.

Example

```
if (MWXSc_Is_DBIIdSCM(symbol) == MWXSuccess) {  
    MWXSc_Get_DBIIdSCM(symbol, &cVal);  
}
```

MWXSc_Get_DoubleSCM

Translates the Scheme double value into an equivalent C value.

Syntax

```
int32 MWXSc_Get_DoubleSCM (  
    MWXSCM_t scm,  
    double *pDouble  
)
```

Parameters

<code>scm</code>	Specifies the Scheme value to translate.
<code>pDouble</code>	Specifies the address in which to store the C double equivalent value of <code>scm</code> .

Return

<code>MWXSuccess</code>	Occurs if the translation is successful.
<code>MWXWrongSymbolType</code>	Occurs if the value is not a double.

Example

```
if (MWXSc_Is_DoubleSCM(scmVal) == MWXSuccess) {  
    MWXSc_Get_DoubleSCM(scmVal, &cVal);  
}
```

MWXSc_Get_IntegerSCM

Translates the Scheme integer value into an equivalent C value.

Syntax

```
int32 MWXSc_Get_IntegerSCM (  
    MWXSCM_t scm,  
    int *pInt  
)
```

Parameters

<code>scm</code>	Specifies the Scheme integer value.
<code>pInt</code>	Specifies the address in which to store C integer equivalent value of <code>scm</code> .

Return

<code>MWXSuccess</code>	Occurs if the translation is successful.
<code>MWXWrongSymbolType</code>	Occurs if the value is not an integer.

Example

```
if (MWXSc_Is_IntegerSCM(scmVal) == MWXSuccess) {  
    MWXSc_Get_IntegerSCM(scmVal, &cVal);  
}
```

MWXSc_Get_StringSCM

Translates the Scheme string value into an equivalent C value. The string pointer refers to memory within the Scheme environment that gets cleaned up regularly by a garbage collector. It is recommended that the string be copied after `MWXSc_Get_StringSCM` finishes to maintain data integrity.

Syntax

```
int32 MWXSc_Get_StringSCM (  
    MWXSCM_t scm,  
    char **pString  
)
```


Parameters

<code>scm</code>	Specifies the Scheme string value.
<code>pString</code>	Specifies the address in which to store the C string equivalent value of <code>scm</code> .

Return

<code>MWXSuccess</code>	Occurs if the translation is successful.
<code>MWXWrongSymbolType</code>	Occurs if the value is not a string.

Example

```
char *temp;
char *cVal = NULL;
if (MWXSc_Is_StringSCM(scmVal) == MWXSuccess) {
    MWXSc_Get_StringSCM(scmVal, &temp);
    cVal = (char*) malloc (strlen(temp) + 1);
    strcpy (cVal, temp);
}
```

MWXSc_Get_Symbol

Gets a C representation for the Scheme symbol with the name `symbolName`.
`MWXSc_Get_Symbol` creates a handle in C that you can use to refer to a variable that is currently active in Scheme.

Syntax

```
MWXSCM_t MWXSc_Get_Symbol (
    char *symbolName
)
```

Parameters

<code>symbolName</code>	Specifies the Scheme symbol to retrieve.

Return

The C representation (handle) of the Scheme symbol.

Example

```
MWXSc_Evaluate("(define c 3)");
MWXSc_Get_IntegerSCM(MWXSc_Get_Symbol("c"), &i);
```

MWXSc_Is_BooleanSCM

Determines if the Scheme data value is Boolean (in Scheme, either #t or #f).

Syntax

```
int32 MWXSc_Is_BooleanSCM (  
    MWXSCM_t scm  
)
```

Parameters

scm	Specifies the Scheme value.

Return

nonzero	Occurs if the value is Boolean.

zero	Occurs if the value is not Boolean.

MWXSc_Is_DoubleSCM

Determines if the Scheme data value is a double value.

Syntax

```
int32 MWXSc_Is_DoubleSCM (  
    MWXSCM_t scm  
)
```

Parameters

scm	Specifies the Scheme value.

Return

nonzero	Occurs if the value is a double value.

zero	Occurs if the value is not a double value.

Example

```
if (MWXSc_Is_DoubleSCM(scmVal) == non-zero) {  
    MWXSc_Get_DoubleSCM(scmVal, &cVal);  
}
```

MWXSc_Is_DBIIdSCM

Determines if the Scheme data value is a database identifier.

Syntax

```
int32 MWXSc_Is_DBIIdSCM (  
    MWXSCM_t scm  
)
```

Parameters

scm	Specifies the Scheme value.

Return

nonzero	Occurs if the value is a database identifier.

zero	Occurs if the value is not a database identifier.

Example

```
if (MWXSc_Is_DBIIdSCM(scmVal) == non-zero) {  
    MWXSc_Get_DBIIdSCM(scmVal, &cVal);  
}
```

MWXSc_Is_IntegerSCM

Determines if the Scheme data value is an integer.

Syntax

```
int32 MWXSc_Is_IntegerSCM (  
    MWXSCM_t scm  
)
```

Parameters

`scm` Specifies the Scheme value.

Return

`nonzero` Occurs if the value is an integer.

`zero` Occurs if the value is not an integer.

Example

```
if (MWXSc_Is_IntegerSCM(scmVal) == non-zero) {  
    MWXSc_Get_IntegerSCM(scmVal, &cVal);  
}
```

MWXSc_Is_ListSCM

Determines if the Scheme data value is a list.

Syntax

```
int32 MWXSc_Is_ListSCM (  
    MWXSCM_t scm  
)
```

Parameters

`scm` Specifies the Scheme value.

Return

`nonzero` Occurs if the value is a list.

`zero` Occurs if the value is not a list.

Example

```
if (MWXSc_Is_ListSCM(scmVal) == non-zero) {  
    for (l = scmVal; ! MWXSc_Is_NullSCM(l); l = MWXSc_Cdr(l)) {  
        scmVal = MWXSc_Car(l);  
        .  
        .  
        .  
    }  
}
```

MWXSc_Is_NullSCM

Determines if a Scheme list is empty.

Syntax

```
MWXSCM_t MWXSc_Is_NullSCM (  
    MWXSCM_t scm  
)
```

Parameters

scm Specifies the Scheme list.

Return

0 Occurs if the Scheme list is not empty; otherwise, returns 1.

Example

```
if (MWXSc_Is_ListSCM(scmVal) == non-zero) {  
    for (l = scmVal; ! MWXSc_Is_NullSCM(l); l = MWXSc_Cdr(l)) {  
        scmVal = MWXSc_Car(l);  
        .  
        .  
        .  
    }  
}
```

MWXSc_Is_StringSCM

Determines if the Scheme data value is a string.

Syntax

```
int32 MWXSc_Is_StringSCM (  
    MWXSCM_t scm  
)
```

Parameters

scm Specifies the Scheme value.

Return

nonzero Occurs if the value is a string.

zero Occurs if the value is not a string.

Example

```
char cVal[256];
if (MWXSc_Is_StringSCM(scmVal) == non-zero) {
    MWXSc_Get_StringSCM(scmVal, &cVal);
}
```

MWXSc_Make_BooleanSCM

Translates a C integer value into a Scheme equivalent Boolean value. A zero translates to false, any other value translates to true.

Syntax

```
MWXSCM_t MWXSc_Make_BooleanSCM (
    int value
)
```

Parameters

value Specifies the C value to translate a Scheme equivalent.

Return

Scheme equivalent Boolean value.

Example

```
scmVal = MWXSc_Make_BooleanSCM(cVal);
```

MWXSc_Make_DBIdSCM

Translates a C database identifier value into a Scheme equivalent database identifier.

Syntax

```
MWXSCM_t MWXSc_Make_DBIdSCM (
    MWXDBId_t value
)
```

Parameters

value Specifies the C value to translate into a Scheme equivalent.

Return

Scheme equivalent database identifier.

Example

```
scmVal = MWXSc_Make_StringSCM(cVal);
```

MWXSc_Make_DoubleSCM

Translates a C double value into a Scheme equivalent double value.

Syntax

```
MWXSCM_t MWXSc_Make_DoubleSCM (  
    double value  
)
```

Parameters

value Specifies the C value to translate a into Scheme equivalent.

Return

Scheme equivalent double value.

Example

```
scmVal = MWXSc_Make_DoubleSCM(cVal);
```

MWXSc_Make_IntegerSCM

Translates a C integer value into a Scheme equivalent integer value.

Syntax

```
MWXSCM_t MWXSc_Make_IntegerSCM (  
    int value  
)
```

Parameters

value Specifies the C value to translate into a Scheme equivalent.

Return

Scheme equivalent integer value.

Example

```
scmVal = MWXSc_Make_IntegerSCM(cVal);
```

MWXSc_Make_ListSCM

Creates a Scheme list from an array of Scheme objects.

Syntax

```
MWXSCM_t MWXSc_Make_ListSCM (
    int    numScm,
    MWXSCM_t scmArr[]
)
```

Parameters

numScm Specifies the number of elements contained in the array from which to construct the list.

scmArr Specifies the array of scheme objects to put into the Scheme list.

Return

A scheme variable containing the Scheme list.

Example

```
int myInt;
    int myBool;
    double myDouble;
    char* myString;
    MWXObjId_t myDBId;
    MWXSCM_t myStuff[5];
    MWXSCM_t myList;
    .
    .
    .
    myStuff[0] = MWXSc_Make_IntegerSCM(myInt);
    myStuff[1] = MWXSc_Make_BooleanSCM(myBool);
    myStuff[2] = MWXSc_Make_DoubleSCM(myDouble);
    myStuff[3] = MWXSc_Make_StringSCM(myString);
    myStuff[4] = MWXSc_Make_DBIdSCM(myDBId);
    myList = MWXSc_Make_ListSCM (5, myStuff);
```

MWXSc_Make_StringSCM

Translates a C string value into a Scheme equivalent string value.

Syntax

```
MWXSCM_t MWXSc_Make_StringSCM (  
    char* value  
)
```

Parameters

value Specifies the C value to translate into a Scheme equivalent.

Return

Scheme equivalent string value.

Example

```
scmVal = MWXSc_Make_StringSCM(cVal);
```

MWXSc_Register_Function

Assigns a Scheme command name to the C procedure that allows the procedure to be called from the Scheme interpreter.

Syntax

```
int32 MWXSc_Register_Function(  
    char *funcName,  
    MWXSCM_t (*func) (MWXSCM_t)  
)
```

Parameters

<code>funcName</code>	Specifies the command name in Scheme that will call the C procedure.
<code>func</code>	Specifies the name of the C procedure being registered.

Return

<code>MWXSuccess</code>	Occurs if the command is registered.
<code>MWXNoLicense</code>	Occurs if no license is available.

Example

```
MWXSCM_t myCommand (MWXSCM_t list) {  
    .  
    .  
    .  
}  
MWXSc_Register_Function("scmCommand", myCommand);
```

MWXSc_Scanf

The `MWXSc_Scanf` works similarly to the C `scanf` command, with one significant variation. Where the `scanf` format string can contain text, the `MWXSc_Scanf` format string contains only format characters. The format characters can be any of the following:

- `b` - Boolean
- `d` - integer
- `f` - double
- `s` - string
- `o` - objId
- `l` - list

Syntax

```
int32 MWXSc_Scanf(  
    MWXSCM_t list,  
    char *formatString, ...  
)
```

Parameters

<code>list</code>	Specifies a set of Scheme data items.
<code>formatString</code>	Specifies a string of characters representing the number, types, and order of Scheme data items to extract from list.

Return

Number of data items extracted from list.

Example

```
MWXSCM_t myCommand (MWXSCM_t list) {  
    int      myBool;  
    int      myInt;  
    double   myDbl;  
    char     myStr;  
    SCM      myList;  
    dbObjId_t myObj;  
    int      count;  
    count = MWXSc_Scanf(list,  
                        "bdfslo",  
                        &myBool, &myInt, &myDbl, &myStr, &myList,  
                        &myObj);  
    .  
    .  
    .  
}
```


8

Place and Route Preparation Functions

This chapter describes the Milkyway API (`MWxDB_*`) database place and route preparation functions. Use these functions to create processes that perform operations similar to those the Scheme commands perform.

MWXDb_Add_InstGroup_ToScanChain

Adds a scan cell instance group to a scan chain in the open cell.

Syntax

```
int MWXDb_Add_InstGroup_ToScanChain (  
    MWXCellId_t cellId, /* INPUT */  
    char *scanChainName, /* INPUT */  
    uint32 numOfItems, /* INPUT */  
    MWXObjId_t *cellInstIds, /* INPUT */  
    char *instGroupName /* INPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell that owns the scan chain, group instance, and cell instances.
scanChainName	Specifies the name of the scan chain.
numOfItems	Specifies the number of cell instances that will be in the group.
cellInstIds	Specifies the name of the array that lists the database identifier of each cell instance that will be added to the group.
instGroupName	Specifies the name of the group to be added to the scan chain.

Return

MWXNoLicense	Occurs if no software license is available.
MWXSuccess	Occurs if the group instance is successfully added to the scan chain.
MWXFail	Occurs if the group instance cannot be successfully added to the scan chain.

Example

```
rc = MWXDb_Add_InstGroup_ToScanChain(cellId, scanChainName, numOfItems,  
cellInstIds, instGroupName);  
  
    switch (rc) {  
        case MWXSuccess:  
            return (MWX_TRUE_SCM);  
        case MWXFail:  
            return (MWX_FALSE_SCM);  
    }
```

Scheme Equivalent

dbAddInstGroupToScanChain (see the *IC Layout Command Reference Manual*)

MWXdB_Add_PartialScan_ToScanChain

Adds specified partial scan cell instances to a scan chain.

Syntax

```
int MWXdB_Add_PartialScan_ToScanChain (
    MWXCellId_t cellId, /* INPUT */
    char *scanChainName, /* INPUT */
    uint32 numItems, /* INPUT */
    MWXObjId_t *cellInstId /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell containing the scan chain.
scanChainName	Name of the scan chain to which partial scan instances will be added.
numItems	Number of cell instances that will be added to the scan chain.
cellInstId	Specifies an array of database identifiers of cell instances that will be added to scan chain.

Return

MWXSuccess	Occurs if the scan cell instances are successfully added to the scan chain.
MWXFail	Occurs if the cell instances are not successfully added to the scan chain.

Example

```
rc = MWXdB_Add_PartialScan_ToScanChain( cellId,
scanChainName, Idcount, Idarray );

switch (rc) {
    case MWXSuccess:
        return (MWX_TRUE_SCM);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to add instances
to scan chain\n");
}
```

```

        return (MWX_FALSE_SCM);
default:
    MWX_Printf("ERROR -- unknown return code from
               MWXDb_Add_PartialScan_ToScanChain\n");
    return (MWX_FALSE_SCM);
}

```

Scheme Equivalent

`dbAddPartialScanToScanChain` (see the *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Assign_VarRouteRule_ById

Assigns a variable-width routing rule set to the net with the specified `netId`.

Syntax

```

int MWXDb_Assign_VarRouteRule_ById (
    MWXCellId_t cellId, /*INPUT*/
    MWXObjId_t netId,   /*INPUT*/
    char *ruleName      /*INPUT*/
)

```


Parameters

<code>cellId</code>	Specifies the database identifier of the cell for the net in which you want to assign the variable-width routing rule.
<code>netId</code>	Specifies the database identifier of the net to assign the rule set.
<code>ruleName</code>	Specifies the name of the rule set you want this rule to be defined in. If the rule set has not already been defined, this function will define it.

Return

<code>MWXSuccess</code>	Occurs if the rule is successfully assigned to the net.
<code>MWXNoLibraryOpened</code>	Occurs if there is no library open.
<code>MWXInvalidCellId</code>	Occurs if the specified cell is 0.
<code>MWXFail</code>	Occurs if the rule set could not be assigned to the net for any other reason.

Example

```
rc = MWXDb_Assign_VarRouteRule_ById (cellId, netId, ruleName);

if (rc != MWXSuccess) {
    MWX_Printf ("ERROR -- Failed to assign routing rule by Id!\n");
    return (MWX_FALSE_SCM);
}

return (MWX_TRUE_SCM);
```

Scheme Equivalent

`dbAssignVarRouteRuleById`

MWXDb_Assign_VarRouteRule_ByNetName

Assigns a variable-width routing rule set to one or more nets. To assign the rule set to multiple nets, set the pattern flag to one and use regular expressions for `netName`.

Syntax

```
int MWXDb_Assign_VarRouteRule_ByNetName (
    MWXCellId_t cellId, /*INPUT*/
    char *ruleName, /*INPUT*/
```

`MWXDb_Assign_VarRouteRule_ByNetName`

```

        int pattern, /*INPUT*/
        char *netName /*INPUT*/
    )

```

Parameters

cellId	Specifies the database identifier of the cell for the net(s) in which you want to assign the variable-width routing rule.
ruleName	Specifies the name of the rule set you want this rule to be defined in. If the rule set has not already been defined, this function will define it.
pattern	Specifies whether <code>netName</code> is a regular expression or a literal expression. If <code>pattern</code> is set to zero, the literal <code>netName</code> is used.
netName	Specifies the name of the net(s) to be assigned the rule set. If the <code>pattern</code> flag is set to one, <code>netName</code> will be treated as a regular expression. For example, the string "NOR.*" will specify all nets beginning with string "NOR".

Return

MWXSuccess	Occurs if the rule is successfully assigned to the specified net(s).
MWXInvalidCellId	Occurs if the specified cell is 0.
MWXFail	Occurs if the rule set could not be assigned to the nets for any other reason.

Example

```

rc = MWXDb_Assign_VarRouteRule_ByNetName (cellId, ruleName, pattern, netName);

if (rc!=MWXSuccess) {
    MWX_Printf ("ERROR -- Failed to assign routing rule to net!\n");
    return (MWX_FALSE_SCM);
}

return (MWX_TRUE_SCM);

```

Scheme Equivalent

```

dbAssignNetVarRouteRule
dbAssignVarRouteRule

```

MWXDb_Attach_ChildCellInstToHierCellInst

Attaches a flattened child-cell instance to a hierarchical cell instance.

Syntax

```
int MWXDb_Attach_ChildCellInstToHierCellInst (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t childHierCellInstId, /* INPUT */
    MWXObjId_t hierCellInstId /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that is opened.
childHierCellInstId	Specifies the database identifier of the flattened child-cell child instance that you want to attach to the specified hierarchical cell instance.
hierCellInstId	Specifies the database identifier of the hierarchical cell instance from which you want to attach a flattened child-cell child instance.

Return

MWXSuccess	Occurs if a flattened child-cell instance is successfully attached to a hierarchical cell instance.
MWXFail	Occurs if a flattened child-cell instance is not successfully attached to a hierarchical cell instance.

Example

```
int attachChildHierCellInstToHierCellInst(
    MWXCellId_t cellId,
    MWXObjId_t childCellInstId,
    MWXObjId_t hierCellInstId
)
{
    int rc;

    rc = MWXDb_Attach_ChildCellInstToHierCellInst(cellId, childCellInstId,
                                                    hierCellInstId);

    if (rc != MWXSuccess) {
        MWX_Printf((char*)"Fail to attach childCellInstance to HierCellInstance\n");
        return 0;
    }
    return 1;
}
```

See Also

[MWXDb_Attach_ChildCellInstToHierCellInst](#)

MWXDb_Attach_ChildHierCellInstToHierCellInst

Attaches a hierarchical child-cell instance to a hierarchical cell instance.

Syntax

```
int MWXDb_Attach_ChildHierCellInstToHierCellInst (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t childHierCellInstId, /* INPUT */
    MWXObjId_t hierCellInstId /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that is opened.
childHierCellInstId	Specifies the database identifier of the hierarchical child-cell instance that you want to attach to the hierarchical cell instance.
hierCellInstId	Specifies the database identifier of the hierarchical child-cell instance from which you want to attach a child hierarchical cell instance.

Return

MWXSuccess	Occurs if a hierarchical child-cell instance is successfully attached.
MWXFail	Occurs if a hierarchical child-cell instance is not successfully attached.

Example

```
int attachChildHierCellInstToHierCellInst(
    MWXCellId_t    cellId,
    MWXObjId_t     childHierCellInstId,
    MWXObjId_t     hierCellInstId
)
{
    int rc;

    rc = MWXDb_Attach_ChildHierCellInstToHierCellInst(cellId, childHierCellInstId,
                                                         hierCellInstId);

    if (rc != MWXSuccess) {
        MWX_Printf((char*)"Fail to attach childHierCellInstance to
HierCellInstance\n");
    }
}
```

```

    return 0;
}
return 1;
}

```

See Also

[MWXDb_Attach_ChildCellInstToHierCellInst](#)

MWXDb_Attach_HierPortInstMasterToHierCellInstMaster

Attaches a hierarchical port instance master to a hierarchical cell instance master.

Syntax

```

int MWXDb_Attach_HierPortInstMasterToHierCellInstMaster (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t hierPortInstMasterId, /* INPUT */
    MWXObjId_t hierCellMasterId /* INPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell that is opened.
hierPortInstMasterId	Specifies the database identifier of the hierarchical port instance master that you want to attach to the hierarchical cell instance master.
hierCellMasterId	Specifies the database identifier of the hierarchical cell instance master to which you want to attach a hierarchical port instance master.

Return

MWXSuccess	Occurs if a hierarchical port instance master is successfully attached to a hierarchical cell instance master.
MWXFail	Occurs if a hierarchical port instance master is not successfully attached to a hierarchical cell instance master.

Example

```

int attachHierPortInstMasterToHierCellInstMaster(
    MWXCellId_t    cellId,
    MWXObjId_t     hierPortInstMasterId,
    MWXObjId_t     hierCellInstMasterId
)
{

```

```

int rc;

rc = MWXDb_Attach_HierPortInstMasterToHierCellInstMaster(cellId,
hierPortInstMasterId,
                                hierCellInstMasterId);

if (rc != MWXSuccess) {
    MWX_Printf((char*)"Fail to attach HierPortInstMaster To
HierCellInstMaster\n");
    return 0;
}
return 1;
}

```

See Also

[MWXDb_Attach_HierPortInstToHierNet](#)

MWXDb_Attach_HierPortInstToHierCellInst

Attaches a hierarchical port instance to a hierarchical cell instance.

Syntax

```

int MWXDb_Attach_HierPortInstToHierCellInst (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t hierPortInstId, /* INPUT */
    MWXObjId_t hierCellInstId /* INPUT */
)

```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell that is opened.
<code>hierPortInstId</code>	Specifies the database identifier of the hierarchical port instance that you want to attach to the hierarchical cell instance.
<code>hierCellInstId</code>	Specifies the database identifier of the hierarchical cell instance from which you want to attach a hierarchical port instance.

Return

<code>MWXSuccess</code>	Occurs if the hierarchical port instance is successfully attached.
<code>MWXFail</code>	Occurs if the hierarchical port instance is not successfully attached.

Example

```
int attachHierPortInstToHierCellInst(
    MWXCellId_t    cellId,
    MWXObjId_t     hierPortInstId,
    MWXObjId_t     hierCellInstId
)
{
    int rc;

    rc = MWXDb_Attach_HierPortInstToHierCellInst(cellId, hierPortInstId,
                                                  hierCellInstId);

    if (rc != MWXSuccess) {
        MWX_Printf((char*)"Fail to attach hierPortInst to HierCellInst\n");
        return 0;
    }
    return 1;
}
```

See Also

[MWXDb_Attach_HierPortInstMasterToHierCellInstMaster](#)

MWXDb_Attach_HierPortInstToHierNet

Attaches a hierarchical port instance to a hierarchical net.

Syntax

```
int MWXDb_Attach_HierPortInstToHierNet (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t hierPortInstId, /* INPUT */
    MWXObjId_t hierNetId /* INPUT */
    uint32 connectType /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that is opened.
hierPortInstId	Specifies the database identifier of the hierarchical port instance that you want to attach to the hierarchical net.
hierNetId	Specifies the database identifier of the hierarchical net from which you want to attach a hierarchical port instance.
connectType	Specifies that you want to connect a hierarchical port instance to a hierarchical net as internally or externally connected. Valid value is either 0 for internal connection or 1 for external connection.

Return

MWXSuccess	Occurs if a hierarchical port instance is successfully attached to a hierarchical net.
MWXFail	Occurs if a hierarchical port instance is not successfully attached to a hierarchical net.

Example

```
int attachHierPortInstToHierNet(
    MWXCellId_t cellId,
    MWXObjId_t hierPortInstId,
    MWXObjId_t hierNetId
)
{
    int rc;

    rc = MWXDb_Attach_HierPortInstToHierNet(cellId, hierPortInstId,
                                              hierNetId);
    if (rc != MWXSuccess) {
        MWX_Printf((char*)"Fail to attach HierPortInst To HierNet\n");
        return 0;
    }
    return 1;
}
```

See Also

[MWXDb_Attach_PortInstToHierNet](#)

MWXdB_Attach_PortInstToHierNet

Attaches a port instance to a hierarchical net.

Syntax

```
int MWXdB_Attach_PortInstToHierNet (  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t portInstId, /* INPUT */  
    MWXObjId_t hierNetId /* INPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell that was opened.
portInstId	Specifies the database identifier of the port instance that you want to attach to the specified hierarchical net.
hierNetId	Specifies the database identifier of the hierarchical net from which you want to attach a port instance.

Return

MWXSuccess	Occurs if a port instance is successfully attached to a hierarchical net.
MWXFail	Occurs if a port instance is not successfully attached to a hierarchical net.

Example

```
int attachPortInstToHierNet(  
    MWXCellId_t cellId,  
    MWXObjId_t portInstId,  
    MWXObjId_t hierNetId  
)  
{  
    int rc;  
  
    rc = MWXdB_Attach_PortInstToHierNet(cellId, portInstId,  
                                         hierNetId);  
  
    if (rc != MWXSuccess) {  
        MWX_Printf((char*)"Fail to attach PortInst To HierNet\n");  
        return 0;  
    }  
    return 1;  
}
```

See Also

[MWXdB_Attach_HierPortInstToHierNet](#)

MWXDb_Clear_CellEEQClass

Clears the electrical-equivalent class specification for the cells in the open library.

Syntax

```
int MWXDb_Clear_CellEEQClass ( )
```

Parameters

None

Return

MWXNoLibraryOpened	Occurs if no library is open.
MWXFail	Occurs if the electrical-equivalent class is not successfully cleared.
MWXSuccess	Occurs if the electrical-equivalent class is successfully cleared.

Example

```
rc = MWXDb_Clear_CellEEQClass();

switch (rc) {
    case MWXSuccess:
        return (MWX_TRUE_SCM);
    case MWXNoLibraryOpened:
        MWX_Printf("ERROR -- No open library was
        found\n");
        return (MWX_FALSE_SCM);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to clear cell
        EEQClass\n");
        return (MWX_FALSE_SCM);
    default:
        MWX_Printf("ERROR -- unknown return code from
        MWXDb_Clear_CellEEQClass\n");
        return (MWX_FALSE_SCM);
}
```

Scheme Equivalent

dbClearCellEEQClass (see the *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Clear_CellEQClass

Clears the electrical-equivalent and logical-equivalent class specification for cells in the open library.

Syntax

```
int MWXDb_Clear_CellEQClass ( )
```

Parameters

None

Return

MWXNoLibraryOpened	Occurs if no library is open.
MWXFail	Occurs if the equivalent class specifications are not successfully cleared.
MWXSuccess	Occurs if the equivalent class specifications are successfully cleared.

Example

```
rc = MWXDb_Clear_CellEQClass();

switch (rc) {
    case MWXSuccess:
        return (MWX_TRUE_SCM);
    case MWXNoLibraryOpened:
        MWX_Printf("ERROR -- No open library was
        found\n");
        return (MWX_FALSE_SCM);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to clear cell
        EQClass\n");
        return (MWX_FALSE_SCM);
    default:
        MWX_Printf("ERROR -- unknown return code from
        MWXDb_Clear_CellEQClass\n");
        return (MWX_FALSE_SCM);
}
```

Scheme Equivalent

dbClearCellEQClass (see the *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Clear_CellLEQClass

Clears the logical-equivalent class specification for the cells in the open library.

Syntax

```
int MWXDb_Clear_CellLEQClass ( )
```

Parameters

None

Return

MWXNoLibraryOpened	Occurs if no library is open.
MWXFail	Occurs if the logical-equivalent class is not successfully cleared.
MWXSuccess	Occurs if the logical-equivalent class is successfully cleared.

Example

```
rc = MWXDb_Clear_CellLEQClass();

switch (rc) {
    case MWXSuccess:
        return (MWX_TRUE_SCM);
    case MWXNoLibraryOpened:
        MWX_Printf("ERROR -- No open library was
        found\n");
        return (MWX_FALSE_SCM);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to clear cell
        LEQClass\n");
        return (MWX_FALSE_SCM);
    default:
        MWX_Printf("ERROR -- unknown return code from
        MWXDb_Clear_CellLEQClass\n");
        return (MWX_FALSE_SCM);
}
```

Scheme Equivalent

`dbClearCellLEQClass` (see the *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Create_BaseArray

Creates a base array record that contains position, class name, tile width, tile height, and its database identifier.

Syntax

```
int MWXDb_Create_BaseArray (
    MWXCellId_t cellId, /* INPUT */
    MWXPoint_t pointArray[], /* INPUT: two points */
    char *className, /* INPUT */
    int32 tileWidth, /* INPUT */
    int32 tileHeight, /* INPUT */
    MWXObjId_t *baseArrayId /* OUTPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell.
pointArray	Specifies the coordinates of the bounding box. The first coordinate indicates the lower left corner of the box; the second indicates the upper right corner of the box.
className	The name of the class.
tileWidth	The tile dimension of the width.
tileHeight	The tile dimension of the height.
baseArrayId	Specifies the database identifier of the base array.

Return

MWXNoLibraryOpened	Occurs if the library is not open.
MWXFail	Occurs if the base array fails to be successfully created.
MWXSuccess	Occurs if the base array is successfully created.

Example

```
rc = MWXDb_Create_BaseArray( cellId, pointArray, className, tileWidth,
    tileHeight, &baseArrayId );

    switch (rc) {
        case MWXSuccess:
            return (MWX_True_SCM)
        case MWXFail:
            MWX_Printf("ERROR -- Failed to operate. \n");
            return (MWX_FALSE_SCM);
```

```
}
```

Scheme Equivalent

dbCreateBaseArray (see the *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Create_BaseArrayRow

Creates a row in the baseArray object.

Syntax

```
int MWXDb_Create_BaseArrayRow (
    MWXCellId_t cellId, /* INPUT */
    MWXPoint_t pointArray[], /* INPUT: two points */
    MWXObjId_t baseArrayId, /* INPUT */
    MWXObjId_t *baseArrayRowId /* OUTPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell.
pointArray	Specifies the row area. The first point indicates the lower corner coordinate; the second point indicates the upper right corner coordinates.
baseArrayId	Specifies the database identifier of the baseArray object.
baseArrayRowId	The database identifier of the base array row.

Return

MWXNoLibraryOpened	Occurs if the library is not open.
MWXFail	Occurs if the base array row cannot be successfully created.
MWXSuccess	Occurs if the base array row is successfully created.

Example

```
rc = MWXDb_Create_BaseArrayRow ( cellId, pointArray,
baseArrayId, &baseArrayRowId);

switch {
    case MWXSuccess:
        return (MWX_True_SCM)
    case MWXFail:
```

```

        MWX_Printf("ERROR -- Failed to operate. \n");
        return (MWX_FALSE_SCM);
    }

```

Scheme Equivalent

dbCreateBaseArrayRow (see the *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Create_CellEEQClass

Establishes the electrical-equivalent class for specified cells in the open library.

Syntax

```

int MWXDb_Create_CellEEQClass (
    char *cellName1, /* INPUT */
    char *cellName2 /* INPUT */
)

```

Parameters

cellName1	Name of the first cell in the electrical-equivalent class cell pair.
cellName2	Name of the second cell in the electrical-equivalent class cell pair.

Return

MWXNoLibraryOpened	Occurs if no library is open.
MWXFail	Occurs if the electrical-equivalent class is not successfully established.
MWXSuccess	Occurs if the electrical-equivalent class is successfully established.

Example

```

rc = MWXDb_Create_CellEEQClass ( cellName1, cellName2 );

switch (rc) {
    case MWXSuccess:
        return (MWX_TRUE_SCM);
    case MWXNoLibraryOpened:
        MWX_Printf("ERROR -- No open library\n");
        return (MWX_FALSE_SCM);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to Create cell
        EEQClass\n");

```

MWXDb_Create_CellEEQClass

```

        return (MWX_FALSE_SCM);
default:
    MWX_Printf("ERROR -- unknown return code from
MWXDb_Create_CellEEQClass\n");
    return (MWX_FALSE_SCM);
}

```

Scheme Equivalent

dbCreateCellEEQClass (see the *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Create_CellInst_IgnoreECO

This function creates a cellInstance object and ignores the ECO.

Syntax

```

int MWXDb_Create_CellInst_IgnoreECO(
    MWXCellId_t cellId /* INPUT */
    char *childLibraryName, /* INPUT */
    char *childCellName, /* INPUT */
    char *cellInstanceName, /* INPUT */
    char *rotationStr, /* INPUT */
    char *mirrorStr, /* INPUT */
    MWXPoint_t *placement, /* INPUT */
    MWXObjId_t *cellInstanceId /* OUTPUT */
)

```


Parameters

cellId	Specifies the cellId in the database
childLibraryName	Specifies the child library name
childCellName	Specifies the child cell Name
cellInstanceName	Specifies the cellInst name that you want to create
rotationStr	Specifies the rotation string
mirrorStr	Specifies the mirror string
placement	Specifies the coordinate of placement of cellInstance
cellInstanceId	Specifies the cellInstance db identifier number

Return

MWXSuccess	Occurs if Milkyway successfully creates a cell instance.
MWXFail	Occurs if Milkyway fails to create a cell Instance.

Example

```
rc = MWXDb_Create_CellInst_IgnoreECO(cellId, childLibraryName, childCellName,
cellInstanceName,
rotationStr, mirrorStr, &placement, &cellInstanceId);
switch (rc) {
    case MWXSuccess:
        return (cellInstanceId);
    case MWXFail:
        printf("Fail to create cellInstance\n");
        break;
    Default:
        printf("Unknown return code \n");
        break;
}
```

See Also

MWXDb_Create_CellInst_WithoutCheckStatus

MWXDb_Create_CellInstance

MWXDb_Create_CellEQClass

Establishes the logical-equivalent class for specified cells in the open library.

MWXDb_Create_CellEQClass

Syntax

```
int MWXDb_Create_CellLEQClass (  
    char *cellName1, /* INPUT */  
    char *cellName2 /* INPUT */  
)
```

Parameters

cellName1	Name of the first cell in the logical-equivalent class cell pair.
cellName2	Name of the second cell in the logical-equivalent class cell pair.

Return

MWXNoLibraryOpened	Occurs if no library is open.
MWXFail	Occurs if the logical-equivalent class is not successfully established.
MWXSuccess	Occurs if the logical-equivalent class is successfully established.

Example

```
rc = MWXDb_Create_CellLEQClass ( cellName1, cellName2 );  
  
switch (rc) {  
    case MWXSuccess:  
        return (MWX_TRUE_SCM);  
    case MWXNoLibraryOpened:  
        MWX_Printf("ERROR -- No open library\n");  
        return (MWX_FALSE_SCM);  
    case MWXFail:  
        MWX_Printf("ERROR -- Failed to Create cell  
LEQClass\n");  
        return (MWX_FALSE_SCM);  
    default:  
        MWX_Printf("ERROR -- unknown return code from  
MWXDb_Create_CellLEQClass\n");  
        return (MWX_FALSE_SCM);  
}
```

Scheme Equivalent

dbCreateCellLEQClass (see the *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Create_Contact_WithRoutType

Creates a contact.

Syntax

```
int MWXDb_Create_Contact_WithRoutType (
    MWXCellId_t cellId, /* INPUT */
    MWXPoint_t *point, /* INPUT */
    uchar contactCodeIndex, /* INPUT */
        /* contactCodeIndex is the same as contact
        number */
    uchar routeType, /* INPUT */
    uchar transformCode, /* INPUT default is 0 (no
        rotation) */
    MWXObjId_t *contactId /* OUTPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell.
point	Specifies the location of the new contact.
contactCodeIndex	The index specified in the technology file that identifies the contact.
routeType	Specifies the route type. Valid data: 0–255.
transformCode	The code of the rotation.
contactId	The database identifier of the contact.

Return

MWXNoLibraryOpened	Occurs if the library is not open.
MWXFail	Occurs if the contact is not successfully created.
MWXSuccess	Occurs if the contact is successfully created.

Example

```
rc = MWXDb_Create_Contact_WithRouteType ( cellId, &point, contactCodeIndex,
routeType, transformCode, &contactId );

switch {
    case MWXSuccess:
        return (MWX_TRUE_SCM)
    case MWXFail:
        MWX_Printf("ERROR -- Failed to operate. \n");
        return (MWX_FALSE_SCM);
}
```

Scheme Equivalent

axCreateContact (see the *IC Layout Command Reference Manual*)

MWXDb_Create_ContactArray_WithRouteType

Creates a contact array, with its route type.

Note:

In `MWXDb_Create_ContactArray_WithRouteType`, the `bitMap` argument is used to represent the contact array in the bitmap.

If the contact array is full, there is no need to use `bitmap`—for example, the `bitmap` is `NULL`. Otherwise, a row-wise bit representation is used to indicate that 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 = size of (uint32) ;
mapSize = ( bitMap ? ((xTimes * yTimes - 1) / (8 * tmp) + 1) * tmp
: 0 ) ;
```

Syntax

```
int MWXDb_Create_ContactArray_WithRoutType (
    MWXCellId_t cellId, /* INPUT */
    MWXPoint_t *point, /* INPUT */
    uchar contactCodeIndex, /* INPUT */
    /* contactCodeIndex is the same as
       contact number */
    uchar routeType, /* INPUT */
    uchar transformCode, /* INPUT default is 0
       (no rotation) */
    int32 xViaPitch, /* INPUT */
    int32 yViaPitch, /* INPUT */
    uint16 xTimes, /* INPUT */
    uint16 yTimes, /* INPUT
       */
    uint32 *bitMap, /* INPUT */
    MWXObjId_t *contactArrayId /* OUTPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell.
point	Specifies the center of the contact.
contactCodeIndex	The index specified in the technology file that indicates the identifier of the contact.
routeType	Specifies the route type.
transformCode	Specifies the rotation code.
xViaPitch	Specifies the space between the contacts, in the X direction.
yViaPitch	Specifies the space between the contacts, in the Y direction.
xTimes	Specifies the repeat times, in the X direction, for creating the contacts.
yTimes	Specifies the repeat times, in the Y direction, for creating the contacts.
bitMap	Specifies the bitmap.
contactArrayId	The database identifier of the contact array.

Return

MWXNoLibraryOpened	Occurs if the library is not open.
MWXFail	Occurs if the contact array cannot be successfully created.
MWXSuccess	Occurs if the contact array is successfully created.

Example

```
rc = MWXDb_Create_ContactArray_WithRouteType ( cellId, &point, contactCodeIndex
routeType,
transformCode, xViaPitch, yViaPitch, xTimes,
yTimes &bitMap, &contactArrayId );

    switch (rc) {
        case MWXSuccess:
            return (MWX_TRUE_SCM)
        case MWXFail:
            MWX_Printf("ERROR -- Failed to operate. \n");
```

```
        return (MWX_FALSE_SCM);  
    }
```

Scheme Equivalent

axCreateContactArray (see the *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Create_CoreBaseArray

This function creates the core base array. The unitTileName needs to be defined in the Tile section of the tech file.

Syntax

```
int MWXDb_Create_CoreBaseArray(  
    MWXcellId_t cellId, /* INPUT */  
    int32 preferDirection, /* INPUT */  
    int32 left, /* INPUT */  
    int32 bottom, /* INPUT */  
    int32 right, /* INPUT */  
    int32 top, /* INPUT */  
    char *unitTileName /* INPUT */  
    MWXObjId_t baseArrayId /* OUTPUT */  
)
```

Parameters

cellId	Specifies the cellId of the cell in the database
preferDirection	Preferred Direction of the core base array
left	Left value of the bounding box
bottom	Bottom value of the bounding box
right	Right value of the bounding box
top	Top value of the bounding box
unitTileName	The default name "unit" will be used if unitTileName is not defined.
baseArrayId	Base Array database identifier number.

Return

MWXSuccess	Occurs if successfully creates the core base array.
MWXFail	Occurs if fails to create core base array.

Example

```
rc = MWXDb_Create_CoreBaseArray (cellId, prferDirection, left, bottom, right,
top, unitTileName,
&baseArrayId);

switch (rc) {
    case MWXSuccess:
        return (MWXSc_Make_DbIdSCM(baseArrayId));
    case MWXFail:
        MWX_Printf("ERROR -- Failed to create base array\n");
        return (MWX_FALSE_SCM);
    default:
        MWX_Printf("ERROR -- unknown return code \n");
        return (MWX_FALSE_SCM);
}
```

Scheme Equivalent

axCreateBaseArrayRecord

MWXDb_Create_Glink

This function creates the global link object.

Syntax

```
int MWXDb_Create_Glink(  
    MWXcellId_t cellId, /* INPUT */  
    MWXObjId_t netId, /* INPUT */  
    int32 layerNumber, /* INPUT */  
    MWXPoint_t *point1, /* INPUT */  
    MWXPoint_t *point2, /* INPUT */  
    MWXObjId_t *glinkId /* OUTPUT */  
)
```

Parameters

cellId	Specifies the cell ID of the cell in the database.
netId	Specifies the ID number of the net.
layerNumberlayer	Number where the global link is going to state.
point1	Coordinates of the first point of the global link.
point2	Coordinates of the second point of the global link.
glinkIdglink	Database identifier number.

Return

MWXSuccess	Occurs if Milkyway successfully creates the global link.
MWXFail	Occurs if Milkyway fails to create the global link.

Example

```
rc = MWXDb_Create_Glink (cellId, netId,layerNumber, point1, point2, &glinkId);  
  
switch (rc) {  
    case MWXSuccess:  
        return (MWXSc_Make_DbIdSCM(glinkId));  
    case MWXFail:  
        MWX_Printf("ERROR -- Failed to create glink\n");  
        return (MWX_FALSE_SCM);  
    default:  
        MWX_Printf("ERROR -- unknown return code \n");  
        return (MWX_FALSE_SCM);  
}
```

MWXDb_Create_HierCellInst

Creates a hierarchical cell instance.

Syntax

```
int MWXDb_Create_HierCellInst (
    MWXCellId_t cellId, /* INPUT */
    char *hierCellInstName, /* INPUT */
    MWXObjId_t hierCellInstMasterId, /* INPUT */
    MWXObjId_t *hierCellInstId /* OUTPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that is opened.
hierCellInstName	Specifies the database identifier of the topmost hierarchical cell instance that is retrieved.
hierCellInstMasterId	Specifies the database identifier of the hierarchical cell instance master to associate to the new cell instance. Use value 0 if the master does not exist.
hierCellInstId	Specifies the database identifier of the hierarchical cell instance that is created.

Return

MWXSuccess	Occurs if the hierarchical cell instance is successfully created.
MWXFail	Occurs if the hierarchical cell instance is not successfully created.

Example

```
MWXObjId_t createHierCellInst(
    MWXCellId_t cellId,
    char *hierCellInstName,
    MWXObjId_t hierCellInstMasterId
)
{
    int rc;
    MWXObjId_t hierCellInstId;

    rc = MWXDb_Create_HierCellInst(cellId, hierCellInstName,
                                    hierCellInstMasterId,
                                    &hierCellInstId);

    if (rc != MWXSuccess) {
        MWX_Printf((char*)"Fail to create hierCellInst \n");
        return 0;
    }
    return hierCellInstId;
}
```

See Also

[MWXDb_Create_HierCellInstMaster](#)

[MWXDb_Set_TopHierCellInst](#)

MWXDb_Create_HierCellInstMaster

Creates a hierarchical cell instance master.

Syntax

```
int MWXDb_Create_HierCellInstMaster (
    MWXCellId_t cellId, /* INPUT */
    char *hierCellInstMasterName, /* INPUT */
    char *hierCellInstMasterId, /* INPUT */
    MWXObjId_t *hierCellInstMasterId /* OUTPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that is opened.
hierCellInstMasterName	Specifies the name of the hierarchical cell instance master to be created.
hierCellInstMasterId	Specifies the database identifier of the hierarchical cell instance master to be created.

Return

MWXSuccess	Occurs if the hierarchical cell instance master is successfully created.
MWXFail	Occurs if the hierarchical cell instance master is not successfully created.

Example

```
MWXObjId_t createHierCellInstMaster (
    MWXCellId_t cellId,
    char *hierCIMName
)
{
    MWXObjId_t hierCIMId;
    int rc;

    rc = MWXDb_Create_HierCellInstMaster(cellId, hierCIMName, &hierCIMId);
    if (rc != MWXSuccess) {
        MWX_Printf((char*) "Fail to create hierCellInstMaster object name %s\n",
                    hierCIMName);
        return 0;
    }
}
```

```

    }
    return hierCIMId;
}

```

See Also

[MWXDb_Create_HierCellInst](#)
[MWXDb_Create_HierPortInstMaster](#)
[MWXDb_Attach_HierPortInstMasterToHierCellInstMaster](#)

MWXDb_Create_HierNet

Creates a hierarchical net.

Syntax

```

int MWXDb_Create_HierCell (
    MWXCellId_t cellId, /* INPUT */
    char *hierNetName, /* INPUT */
    MWXObjId_t *hierNetId /* OUTPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell that is opened.
hierNetName	Specifies the name of the hierarchical net to be created.
hierNetId	Specifies the database identifier of the hierarchical net to be created.

Return

MWXSuccess	Occurs if the hierarchical net is successfully created.
MWXFail	Occurs if the hierarchical net is not successfully created.

Example

```

MWXObjId_t createHierNet(
    MWXCellId_t cellId,
    char *hierNetName
)
{
    int rc;
    MWXObjId_t hierNetId;

    rc = MWXDb_Create_HierNet(cellId, hierNetName, &hierNetId);
    if (rc != MWXSuccess) {
        MWX_Printf((char*) "Fail to create HierNet %s\n",hierNetName);
        return 0;
    }
}

```

```
}  
return hierNetId;  
}
```

See Also

[MWXDb_Create_HierPortInst](#)

[MWXDb_Attach_HierPortInstToHierNet](#)

MWXDb_Create_HierPortInst

Creates a hierarchical port instance.

Syntax

```
int MWXDb_Create_HierCellInstMaster (  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t hierCellInstId, /* INPUT */  
    MWXObjId_t netId1, /* INPUT */  
    MWXObjId_t netId2, /* INPUT */  
    MWXObjId_t hierPortInstMasterId, /* INPUT */  
    MWXObjId_t *hierPortInstId /* OUTPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell that is opened.
hierCellInstId	Specifies the database identifier of the hierarchical cell instance to apply to the new port instance. Valid value can also be 0.
netId1	Specifies the database identifier of the hierarchical net internally connected at the port. Valid value can also be 0.
netId2	Specifies the database identifier of the hierarchical net externally connected at the port. Valid value can also be 0.
hierPortInstMasterId	Specifies the database identifier of the hierarchical port instance master to apply to the newly created port instance.
hierPortInstId	Specifies the database identifier of the hierarchical port instance to be created.

Return

MWXSuccess	Occurs if the hierarchical port instance is successfully created.
MWXFail	Occurs if the hierarchical port instance is not successfully created.

Example

```
MWXObjId_t createHierNet(
    MWXCellId_t cellId,
    char *hierNetName
)
{
    int rc;
    MWXObjId_t hierNetId;

    rc = MWXDb_Create_HierNet(cellId, hierNetName, &hierNetId);
    if (rc != MWXSuccess) {
        MWX_Printf((char*) "Fail to create HierNet %s\n", hierNetName);
        return 0;
    }
    return hierNetId;
}

MWXObjId_t createHierPortInst(
    MWXCellId_t cellId,
    char *hierNetName1,
    char *hierNetName2,
    MWXObjId_t hierCellInstId,
```

```

    MWXObjId_t hierPortInstMasterId
    )
{
    int rc;
    MWXObjId_t hierPortInstId;
    MWXObjId_t hierNetId1, hierNetId2;
    uint32 connectType;

    hierNetId1 = 0;
    hierNetId2 = 0;
    if (hierNetName1) {
        hierNetId1 = createHierNet(cellId, hierNetName1);
        if (hierNetId1 == 0) {
            return 0;
        }
    }
    if (hierNetName2) {
        hierNetId2 = createHierNet(cellId, hierNetName2);
        if (hierNetId2 == 0) {
            return 0;
        }
    }
    rc = MWXDb_Create_HierPortInst(cellId, hierCellInstId, hierNetId1,
                                   hierNetId2, hierPortInstMasterId,
                                   &hierPortInstId);

    if (rc != MWXSuccess) {
        MWX_Printf((char*)"Fail to create hierPortInst \n");
        return 0;
    }
    connectType = 0; /* internal connection */
    if (hierNetId1) {
        rc = MWXDb_Attach_HierPortInstToHierNet(cellId, hierPortInstId,
                                                hierNetId1, connectType);

        if (rc != MWXSuccess) {
            MWX_Printf((char*)"Fail to attach hierPortInst to HierNet\n");
            return 0;
        }
    }
    connectType = 0; /* internal connection */
    if (hierNetId2) {
        rc = MWXDb_Attach_HierPortInstToHierNet(cellId, hierPortInstId,
                                                hierNetId2, connectType);

        if (rc != MWXSuccess) {
            MWX_Printf((char*)"Fail to attach hierPortInst to HierNet\n");
            return 0;
        }
    }
    return hierPortInstId;
}

```

See Also

[MWXDb_Create_HierCellInst](#)
[MWXDb_Create_HierCellInstMaster](#)
[MWXDb_Create_HierPortInstMaster](#)
[MWXDb_Create_HierNet](#)
[MWXDb_Attach_HierPortInstToHierNet](#)

MWXDb_Create_HierPortInstMaster

Creates a hierarchical port instance master.

Syntax

```
int MWXDb_Create_HierPortInstMaster (
    MWXCellId_t cellId, /* INPUT */
    char *hierPortInstMasterName, /* INPUT */
    MWXObjId_t hierCellInstMasterId, /* INPUT */
    unsigned char direction, /* INPUT */
    MWXObjId_t *hierPortInstMasterId /* OUTPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that is opened.
hierPortInstMasterName	Specifies the name of the hierarchical port instance master to be created.
hierCellInstMasterId	Specifies the database identifier of the hierarchical cell instance master to associate with the hierarchical port instance master. Valid value can also be 0.
direction	Specifies the direction of the port instance master.
hierPortInstMasterId	Specifies the database identifier of the hierarchical port instance master to be created.

Return

MWXSuccess	Occurs if the hierarchical port instance master is successfully created.
MWXFail	Occurs if the hierarchical port instance master is not successfully created.

Example

```
MWXObjId_t createHierPortInstMaster (
    MWXCellId_t cellId,
    char *hierPortInstMasterName,
    MWXObjId_t hierCellInstMasterId,
    unsigned char direction
)
{
    MWXObjId_t hierPortInstMasterId;
    int rc;
    rc = MWXDb_Create_HierPortInstMaster(cellId, hierPortInstMasterName,
                                          hierCellInstMasterId, direction,
                                          &hierPortInstMasterId);
}
```



```

if (rc != MWXSuccess) {
    MWX_Printf((char*)"Fail to create hierPortInstMaster object name %s\n",
               hierPortInstMasterName);
    return 0;
}

return hierPortInstMasterId;
}

```

See Also

[MWXDb_Create_HierCellInst](#)
[MWXDb_Create_HierCellInstMaster](#)
[MWXDb_Create_HierPortInst](#)
[MWXDb_Create_HierPortVectorInfo](#)

MWXDb_Create_HierPortVectorInfo

Creates vector information for a hierarchical port instance master.

Syntax

```

int MWXDb_Create_HierPortVectorInfo (
    MWXCellId_t cellId, /* INPUT */
    int32 useLeftBound, /* INPUT */
    int32 useRightBound, /* INPUT */
    int32 decLeftBound, /* INPUT */
    int32 decRightBound, /* INPUT */
    MWXObjId_t hierPortInstMasterId, /* INPUT */
    MWXObjId_t *hierPortVectorInfoId /* OUTPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell that is opened.
useLeftBound	Specifies the left bound used inside the port name argument list.
useRightBound	Specifies the right bound used inside the port name argument list.
decLeftBound	Specifies the left bound in the port and data type declarations.
decRightBound	Specifies the right bound in the port and data type declarations.
hierPortInstMasterId	Specifies the database identifier of the hierarchical port instance master to associate with this port vector information. Valid value can also be 0.
hierPortVectorInfoId	Specifies the database identifier for the newly created port vector information.

Return

MWXSuccess	Occurs if the command successfully creates port vector information.
MWXFail	Occurs if the command fails to create port vector information.

Example

```
MWXObjId_t createHierPortInstVectorInfo (
    MWXCellId_t cellId,
    MWXObjId_t hierPortInstMasterId,
    int32 useLeftBound,
    int32 useRightBound,
    int32 decLeftBound,
    int32 decRightBound
)
{
    int32 rc;
    MWXObjId_t hierPortVectorInfoId;

    rc = MWXDb_Create_HierPortVectorInfo(cellId, useLeftBound, useRightBound,
                                          decLeftBound, decRightBound,
                                          hierPortInstMasterId, &hierPortVectorInfoId);

    if (rc != MWXSuccess) {
        MWX_Printf((char*)"Fail to create hier portVectorInfo\n");
        return 0;
    }
}
```

```

return hierPortVectorInfoId;
}

```

See Also

[MWXDb_Create_HierPortInstMaster](#)

MWXDb_Create_PcellDefinition

This function creates a Pcell (parameterized cell) definition. A Pcell allows parameterized instantiations of basic circuit elements.

Syntax

```

int MWXDb_Create_PcellDefinition(
    MWXlibId_t libId, /* INPUT */
    char *pcellObjName /* INPUT */
    char *srcFile /* INPUT */
)

```

Parameters

libId	Specifies the db identifier number of the current library
pcellObjectName	Specifies the parameterized cell object name
srcFile	Name of the source file

Return

MWXSuccess	Occurs if successfully creates the parameterized cell definition
MWXFail	Occurs if fails to create parameterized cell definition

Example

```

rc = MWXDb_Create_PcellDefinition (libId, pcellObjectName, srcFile);

switch (rc) {
    case MWXSuccess:
        MWX_Printf(Success to create pcellDefintion)
        return (MWX_SUCCESS_SCM);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to create pcellDefintion\n");
        return (MWX_FALSE_SCM);
    default:
        MWX_Printf("ERROR -- unknown return code \n");
        return (MWX_FALSE_SCM);
}

```

See Also

MWXDb_Create_PcellInstance

MWXDb_Delete_PcellDefintion

MWXDb_Create_PcellInstance

This function creates a PcellInstance (parameterized cell instance).

Syntax

```
int MWXDb_Create_PCellInstance (
    MWXCellId_t cellId /* INPUT */
    char *childLibraryName, /* INPUT */
    char *pcellName, /* INPUT */
    char *pcellInstName, /* INPUT */
    char *rotationStr, /* INPUT */
    char *mirrorStr, /* INPUT */
    MWXPoint_t *placement, /* INPUT */
    MWXObjId_t *pcellInstId /* OUTPUT */
)
```

Parameters

cellId	Specifies the cellId in the database
childLibraryName	Specifies the child library name
pcellName	Specifies the name of the p-cell
pcellInstName	Specifies the pcellInst name
rotationStr	Specifies the rotation string
mirrorStr	Specifies the mirror string
placement	Specifies the coordinate of placement of cellInstance
pcellInstanceId	Specifies the cellInstance db identifier number

Return

MWXSuccess	Occurs if successfully create pcell instance.
MWXFail	Occurs if fail to create pcell Instance

Example

```
rc = MWXDb_Create_PCellInstance (cellId, childLibraryName, pcellName,
pcellInstName, rotationStr, mirrorStr, &placement, &pcellInstId);
switch (rc) {
    case MWXSuccess:
        return (pcellInstId);
    case MWXFail:
        printf("Fail to create pcellInstance\n");
        break;
    Default:
        printf("Unknown return code \n");
        break;
}
```

See Also

MWXDb_Create_PcellDefintion

MWXDb_Delete_PcellDefinition

MWXDb_Create_Row

The MWXDb_Create_Row function is similar to the Scheme axCreateRowRecord function.

MWXDb_Create_Row

Syntax

```
int32 MWXDb_Create_Row(  
    MWXCellId_t cellId, /* INPUT */  
    int32 preferDirection, /* INPUT */  
    int32 flip, /* INPUT : 0 means not flipped*/  
    char *patternName, /* INPUT */  
    MWXPoint_t pointArray[], /* INPUT: two points */  
    MWXObjId_t baseArrayId, /* INPUT */  
    MWXObjId_t *baseArrayRowId /* OUTPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell in which to create a rectangle.
preferDirection	Specifies the preferred direction vertical or horizontal.
flip	Specifies whether the tile will be flipped or not. 0 means not flipped.
patternName	The pattern name of the tile you want to create.
pointArray	Specifies the coordinates of two points for creating a cell row.
baseArrayId	Specifies the database identifier of the base array.
baseArrayRowId	Specifies the database identifier of the base array cell row.

Return

MWXSuccess	Occurs if the Row object is successfully created.
MWXFail	Occurs if the Row object is not successfully created.

Example

```
rc = MWXDb_Create_Row (cellId,preferDirection,flip, patternName, pointArray,  
baseArrayId, &rowId);  
switch (rc) {  
    case MWXSuccess:  
        rtn = MWXSc_Make_DBIIdSCM (rowId);  
        break;  
    case MWXFail:  
        rtn = MWX_FALSE_SCM;  
        break;  
    default:  
        rtn = MWX_FALSE_SCM;  
}  
  
return (rtn);  
}
```

Scheme Equivalent

`axCreateRowRecord` (see the *Milkyway Environment Extension Language Reference Manual*)

MWXdB_Create_ScanChain

Creates an empty scan chain in the open cell.

Syntax

```
int MWXdB_Create_ScanChain (
    MWXCellId_t cellId, /* INPUT */
    char *scanChainName /* INPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the open cell.
<code>scanChainName</code>	Specifies the name of the scan chain.

Return

<code>MWXNoLicense</code>	Occurs if no software license is available.
<code>MWXSuccess</code>	Occurs if the scan chain is successfully created.
<code>MWXFail</code>	Occurs if the scan chain cannot be successfully created.

Example

```
rc = MWXdB_Create_ScanChain( cellId, scanChainName );
switch (rc) {
    case MWXSuccess:
        return (MWX_TRUE_SCM);
    case MWXFail:
        return (MWX_FALSE_SCM);
}
```

Scheme Equivalent

`dbCreateScanChain` (see the *IC Layout Command Reference Manual*)

MWXdB_Create_SGEdge

This function creates a SGEdge.

Syntax

```
int MWXDb_Create_SGEdge(  
    MWXCellId_t cellId /* INPUT */  
    int status /* INPUT */  
    MWXObjId_t fromObjId /* INPUT */  
    MWXObjId_t toObjId /* INPUT */  
    MWXObjId_t *SGEdgeId /* OUTPUT */  
)
```

Parameters

cellId	Specifies the cellId in the database
status	status of the SGEdge object
fromObjId	Specifies the db identifier number of the object that we use to create SGEdge from
toObjId	Specifies the db identifier number of the object that we use to create SGEdge to
SGEdgeId	SGEdge new db identifier number

Return

MWXSuccess	Occurs if successfully create SGEdge.
MWXFail	Occurs if fail to create SG Edge

Example

```
rc = MWXDb_Create_SGEdge(cellId, status, fromObjId, toObjId, &SGEdgeId);  
switch (rc) {  
    case MWXSuccess:  
        return (SGEdgeId);  
    case MWXFail:  
        printf("Fail to create SGEdge object\n");  
        break;  
    Default:  
        printf("Unknown return code \n");  
        break;  
}
```

MWXDb_Create_SGPrimaryNode

This function creates a SG primary node.

Syntax

```
int MWXDb_Create_SGPrimaryNode(  
    MWXCellId_t cellId /* INPUT */  
    MWXObjId_t fromObjId /* INPUT */  
    int status /* INPUT */  
    MWXObjId_t toObjId /* INPUT */  
    MWXObjId_t *SGPrimaryNode /* OUTPUT */  
)
```

Parameters

cellId	Specifies the cellId in the database
status	Status of the SG Primary Node object
fromObjId	Specifies the db identifier number of the object that we use to create SG Primary Node
from	
toObjId	Specifies the db identifier number of the object that we use to create SG Primary Node to
SGEdgId	SG Primary Node new db identifier number

Return

MWXSuccess	Occurs if successfully create SG Primary Node.
MWXFail	Occurs if fail to create SG Primary Node

Example

```
rc = MWXDb_Create_SGPrimaryNode(cellId, fromObjId, status , toObjId, &SGEdgeId);  
switch (rc) {  
    case MWXSuccess:  
        return (SGPrimaryNodeId);  
    case MWXFail:  
        printf("Fail to create SG Primary Node object\n");  
        break;  
    Default:  
        printf("Unknown return code \n");  
        break;  
}
```

MWXDb_Create_TilePattern

Creates a tile pattern, where the `unitTileName` should be defined in the tile section of the technology file. The `MWXDb_Create_TilePattern` function is similar to the `axCreateTilePatternRecord` **Scheme** function.

Syntax

```
int32 MWXDb_Create_TilePattern(  
    MWXCellId_t cellId, /* INPUT */  
    int32 preferDirection, /* INPUT */  
    int32 flip, /* INPUT : 0 means not flipped*/  
    char *unitTileName, /* INPUT */  
    char *patternName /* INPUT */  
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell that was opened.
<code>preferDirection</code>	Specifies the preferred direction: vertical or horizontal.
<code>flip</code>	Specifies whether the tile will be flipped or not. A 0 means not flipped.
<code>unitTileName</code>	The unit of the tile. If the unit is not specified—for example, it is equivalent to NULL—the default <code>unitTileName</code> unit will be used.
<code>patternName</code>	The pattern name of the tile you want to create.

Return

<code>MWXSuccess</code>	Occurs if the tile pattern is successfully created.
<code>MWXFail</code>	Occurs if the tile pattern in not successfully created.

Example

```
rc = MWXDb_Create_TilePattern (cellId,preferDirection,flip,unitTileName,
patternName);
switch (rc) {
    case MWXSuccess:
        rtn = MWX_TRUE_SCM;
        break;
    case MWXFail:
        rtn = MWX_FALSE_SCM;
        break;
    default:
        rtn = MWX_FALSE_SCM;
}
return (rtn);
}
```

Scheme Equivalent

`axCreateTilePatternRecord` (see the *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Create_Track

The `MWXDb_Create_Track` function is similar to the `axCreateTrackRecord` Scheme function.

Syntax

```
int32 MWXDb_Create_Track(  
    MWXCellId_t cellId, /* INPUT */  
    int32 preferDirection, /* INPUT */  
    int32 layer, /* INPUT */  
    int32 start, /* INPUT */  
    int32 step, /* INPUT */  
    int32 xxtimes, /* INPUT */  
    MWXPoint_t pointArray[], /* INPUT: two points */  
    char *unitTileName, /* INPUT */  
    MWXObjId_t *trackId /* OUTPUT */  
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell that was opened.
<code>preferDirection</code>	Specifies the preferred direction: vertical or horizontal.
<code>layer</code>	Specifies the layer number of the track.
<code>start</code>	Specifies the start track.
<code>step</code>	Specifies the distance between tracks.
<code>xxtimes</code>	Specifies the number of tracks created.
<code>pointArray</code>	Specifies two points used to create the track.
<code>unitTileName</code>	The unit of the tile. If the unit is not specified—for example, it is set equivalent to NULL—the default <code>unitTileName</code> unit will be used.
<code>trackId</code>	Specifies the database identifier of the track.

Return

<code>MWXSuccess</code>	Occurs if the track record is successfully created.
<code>MWXFail</code>	Occurs if the track record is not successfully created.

Example

```
rc = MWXDb_Create_Track (cellId,preferDirection,layerNum,startTrack,
startStep,count, pointArray, unitTileName, &trackId);
switch (rc) {
    case MWXSuccess:
        rtn = MWXSc_Make_DBIIdSCM (trackId);
        break;
    case MWXFail:
        rtn = MWX_FALSE_SCM;
        break;
    default:
        rtn = MWX_FALSE_SCM;
}
return (rtn);
}
```

Scheme Equivalent

`axCreateTrackRecord` (see the *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Create_Vertical_Wire

Creates a vertical wire in the cell identified by `cellId` at the specified location with the given length.

Syntax

```
int MWXDb_Create_Vertical_Wire(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t wireMasterId, /* INPUT */  
    int32 wireLength, /* INPUT */  
    MWXPoint_t *point, /* INPUT */  
    MWXObjId_t *wireId /* OUTPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell in which to create the vertical wire.
wireMasterId	Specifies the database identifier of the wire master to use for the new wire.
wireLength	Specifies the length of the wire. A positive number extends the wire above the origin; a negative number extends the wire below the origin.
point	Specifies the origin of the wire.
wireId	Specifies the database identifier of the new wire.

Return

MWXSuccess	Occurs if the wire is successfully created.
MWXNoLibraryOpened	Occurs if a library is not open.
MWXFail	Occurs if the wire is not successfully created for any other reason.

Example

```
rc = MWXDb_Create_Vertical_Wire (cellId,\
wireMasterId, wireLength, &point, &wireId);

switch (rc) {
    case MWXSuccess:
        return (MWXSc_Make_DBIIdSCM (wireId));
    default:
        return (MWX_FALSE_SCM);
}
```

Scheme Equivalent

dbCreateVerticalWire (see the *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Create_WireDirection

Creates a wire direction in which the unit tile name should be defined in the tile section of the technology file. The MWXDb_Create_WireDirection function is similar to the axCreateWireDirRecord Scheme function.

Syntax

```
int32 MWXDb_Create_Wire_Direction(  
    MWXCellId_t cellId, /* INPUT */  
    int32 preferDirection, /* INPUT */  
    int32 layer, /* INPUT */  
    int32 *unitTileName, /* INPUT: if not specified, for example, ==  
    NULL, then the default unitTileName "unit" will be used. */  
    MWXObjId_t *wireDirectionId /* OUTPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell that was opened.
preferDirection	Specifies the preferred direction: vertical or horizontal.
layer	Specifies the number of the layer where the wire direction can be created.
unitTileName	The unit of the tile. If not specified—for example, it is set equal to NULL—the default unit tile name unit will be used.
wireDirectionId	Specifies the database identifier of the <code>wireDirection</code> object.

Return

MWXSuccess	Occurs if the wire direction object is successfully created.
MWXFail	Occurs if the wire direction object is not successfully created.

Example

```
rc = MWXDb_Create_WireDirection (cellId,preferDirection,layer,unitTileName,  
&wireDirectionId);  
switch (rc) {  
    case MWXSuccess:  
        rtn = MWXSc_Make_DBIdSCM (wireDirectionId);  
        break;  
    case MWXFail:  
        rtn = MWX_FALSE_SCM;  
        break;  
    default:  
        rtn = MWX_FALSE_SCM;  
}  
return (rtn);  
}
```

Scheme Equivalent

`axCreateWireDirRecord` (see *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Define_VarRouteRule_Contact

Defines a contact rule for variable-width routing in the set `ruleName`.

Syntax

```
int MWXDb_Define_VarRouteRule_Contact (  
    MWXCellId_t cellId, /*INPUT*/  
    char *ruleName, /*INPUT*/  
    char *contactName, /*INPUT*/  
    int32 xTimes, /*INPUT*/  
    int32 yTimes /*INPUT*/  
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell for which you want to define the variable-width routing rule.
<code>ruleName</code>	Specifies the name of the rule set you want this rule to be defined in. If the rule set has not already been defined, this function will define it.
<code>contactName</code>	Name of the contact to which the rule applies.
<code>xTimes</code>	Specifies the number of columns in the contact array.
<code>yTimes</code>	Specifies the number of rows in the contact array.

Return

<code>MWXSuccess</code>	Occurs if the rule was successfully defined.
<code>MWXNoLibraryOpened</code>	Occurs if there is no library open.
<code>MWXInvalidCellId</code>	Occurs if the specified cell is 0.
<code>MWXFail</code>	Occurs if the rule was not defined for any other reason.

Example

```
rc = MWXDb_Define_VarRouteRule_Contact (cellId, ruleName, str, xTimes, yTimes);  
  
switch (rc) {  
    case MWXSuccess:  
        return (MWX_TRUE_SCM);  
    case MWXFail:  
        MWX_Printf("ERROR -- Failed to define.\n");  
        return (MWX_FALSE_SCM);  
    default:  
        MWX_Printf("ERROR -- unknown return code from
```

```

        MWXDb_Define_VarRouteRule_Contact\n");
    return (MWX_FALSE_SCM);
}

```

Scheme Equivalent

dbDefineVarRouteRule

MWXDb_Define_VarRouteRule_Spacing

Defines a spacing or shield-spacing rule for variable-width routing in the set `ruleName`.

Syntax

```

int MWXDb_Define_VarRouteRule_Spacing (
    MWXCellId_t cellId, /*INPUT*/
    char *ruleName, /*INPUT*/
    int32 shieldOrNot, /*INPUT*/
    char *layer, /*INPUT*/
    double value /*INPUT*/
)

```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell for which you want to define the variable-width routing rule.
<code>ruleName</code>	Specifies the name of the rule set you want this rule to be defined in. If the rule set has not already been defined, this function will define it.
<code>shieldOrNot</code>	Specifies whether this is a spacing rule or a shield spacing rule. If <code>shieldOrNot</code> is set to zero, then a spacing rule is defined.
<code>layer</code>	Specifies the name of the layer to which the rule applies.
<code>value</code>	Specifies the value to define.

Return

<code>MWXSucccess</code>	Occurs if the rule is successfully defined.
<code>MWXNoLibraryOpened</code>	Occurs if there is no library open.
<code>MWXInvalidCellId</code>	Occurs if the specified cell is 0.
<code>MWXFail</code>	Occurs if the rule was not defined for any other reason.

Example

```
rc = MWXDb_Define_VarRouteRule_Spacing (cellId, ruleName, flag, str, doubleVal);

switch (rc) {
    case MWXSucccess:
        return (MWX_TRUE_SCM);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to define.\n");
        return (MWX_FALSE_SCM);
    default:
        MWX_Printf("ERROR -- unknown return code from
                    MWXDb_Define_VarRouteRule_Spacing\n");
        return (MWX_FALSE_SCM);
}
```

Scheme Equivalent

`dbDefineVarRouteRule`

MWXdB_Define_VarRouteRule_TaperLevel

Defines a taper level or a shield rule for variable-width routing in the set `ruleName`.

Syntax

```
int MWXdB_Define_VarRouteRule_TaperLevel (
    MWXCellId_t cellId, /*INPUT*/
    char *ruleName, /*INPUT*/
    int32 shieldOrNot, /*INPUT*/
    int32 value /*INPUT*/
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell for which you want to define the variable-width routing rule.
<code>ruleName</code>	Specifies the name of the rule set you want this rule to be defined in. If the rule set has not already been defined, this function will define it.
<code>shieldOrNot</code>	Specifies whether this is a taper-level rule or a shield rule. If <code>shieldOrNot</code> is set to zero, then a taper-level rule is defined.
<code>value</code>	Specifies the value to define.

Return

<code>MWXSuccess</code>	Occurs if the rule is successfully defined.
<code>MWXNoLibraryOpened</code>	Occurs if there is no library open.
<code>MWXInvalidCellId</code>	Occurs if the specified cell is 0.
<code>MWXFail</code>	Occurs if the rule was not defined for any other reason.

Example

```
rc = MWXdB_Define_VarRouteRule_TaperLevel (cellId, ruleName, flag, value);

switch (rc) {
    case MWXSuccess:
        return (MWX_TRUE_SCM);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to operate.\n");
        return (MWX_FALSE_SCM);
    default:
        MWX_Printf("ERROR -- unknown return code from
                    MWXdB_Define_VarRouteRule_TaperLevel\n");
        return (MWX_FALSE_SCM);
}
```

```
}
```

Scheme Equivalent

dbDefineVarRouteRule

MWXdB_Define_VarRouteRule_Width

Defines a width or shield width for variable-width routing in the set `ruleName`.

Syntax

```
int MWXdB_Define_VarRouteRule_Width (  
    MWXCellId_t cellId, /*INPUT*/  
    char *ruleName, /*INPUT*/  
    int32 shieldOrNot, /*INPUT*/  
    char *layer, /*INPUT*/  
    double value /*INPUT*/  
)
```

Parameters

cellId	Specifies the database identifier of the cell for which you want to define the variable-width routing rule.
ruleName	Specifies the name of the rule set you want this rule to be defined in. If the rule set has not already been defined, this function will define it.
shieldOrNot	Specifies whether this is a width rule or a shield-width rule. If <code>shieldOrNot</code> is set to zero, then a width rule is defined.
layer	Specifies the name of the layer to which the rule applies.
value	Specifies the value to define.

Return

MWXSuccess	Occurs if the rule is successfully defined.
MWXNoLibraryOpened	Occurs if there is no library open.
MWXInvalidCellId	Occurs if the specified cell is 0.
MWXFail	Occurs if the rule was not defined for any other reason.

Example

```
rc = MWXDb_Define_VarRouteRule_Width (cellId, ruleName, flag, str, doubleVal);

switch (rc) {
    case MWXSuccess:
        return (MWX_TRUE_SCM);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to define.\n");
        return (MWX_FALSE_SCM);
    default:
        MWX_Printf("ERROR -- unknown return code from
                    MWXDb_Define_VarRouteRule_Width\n");
        return (MWX_FALSE_SCM);
}
```

Scheme Equivalent

dbDefineVarRouteRule

MWXdB_Delete_AllRouteGuides

Delete all the route guide database identifiers for the given cell.

Syntax

```
int MWXdB_Delete_AllRouteGuides (
    MWXCellId_t  cellId      /* INPUT */
)
```

Parameter

cellId	Specifies the database identifier of the cell.

Return

MWXSuccess	Occurs if the function is successful.

MWXFail	Occurs if the function is unsuccessful.

Example

```
int deleteAllRouteGuides( MWXCellId_t cellId) {
    if (MWXdB_Delete_AllRouteGuides( cellId) != MWXSuccess) {
        MWX_Printf("failed to delete all routeGuide objects\n");
        return (MWXFail);
    }
    MWX_Printf("All RouteGuides have been deleted\n");
    return (MWXSuccess);
}
```

MWXdB_Delete_AllVarRouteRules

Deletes all the variable-width routing rules in the specified cell.

Syntax

```
int MWXdB_Delete_AllVarRouteRules (
    MWXCellId_t cellId /*INPUT*/
)
```

Parameters

cellId	Specifies the database identifier of the cell in which you want to delete all the variable-width routing rules.

Return

MWXSuccess	Occurs if the variable-width routing rules are deleted successfully.
MWXNoLibraryOpened	Occurs if there is no library open.
MWXInvalidCellId	Occurs if the specified cell is 0.
MWXFail	Occurs if the variable-width routing rules were not deleted successfully.

Example

```
rc = MWXDb_Delete_AllVarRouteRules (cellId);

switch (rc) {
    case MWXSuccess:
        return (MWX_TRUE_SCM);
    case MWXFail:
        MWX_Printf ("ERROR -- Failed to delete variable-width routing rules!\n");
        return (MWX_FALSE_SCM);
    default:
        MWX_Printf("ERROR -- unknown return code from
                    MWXDb_Delete_AllVarRouteRules\n");
        return (MWX_FALSE_SCM);
}
```

Scheme Equivalent

dbDeleteAllVarRouteRules

MWXDb_Delete_CellInstance_FromCellRegion

Deletes a cell instance's association with a cell region. The cell instance will not change its location. Only the relationship indicating that this cell instance is to be placed inside the cell region is removed.

Syntax

```
int MWXDb_Delete_CellInstance_FromCellRegion (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t cellRegionId, /* INPUT */
    MWXObjId_t cellInstanceId /* INPUT */
)
```


Parameters

<code>cellId</code>	Specifies the database identifier of the cell.
<code>cellRegionId</code>	Specifies the database identifier of the region to which the cell instance information applies.
<code>cellInstanceId</code>	Specifies the database identifier of the cell instance.

Return

<code>MWXSuccess</code>	Occurs if the association of the cell instance with the cell region is successfully removed.
<code>MWXFail</code>	Occurs if the association of the cell instance with the cell region cannot be successfully removed.

Example

```
rc = MWXDb_Delete_CellInstance_FromCellRegion ( cellId, cellRegionId,
cellInstanceId );

switch (rc) {
    case MWXSuccess:
        return (MWX_True_SCM)
    case MWXFail:
        MWX_Printf("ERROR -- Failed to operate. \n");
        return (MWX_FALSE_SCM);
}
```

Scheme Equivalent

`dbDeleteCellInstFromCellRegion` (see the *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Delete_PcellDefinition

This function deletes the Pcell (parameterized cell) definition from a specified library.

Syntax

```
int MWXDb_Delete_PcellDefinition(
MWXLibId_tlibId /* INPUT */
char *pcellObjName /* INPUT */
)
```

Parameters

libId	Specifies the db identifier of the current library
pcellObjName	Specifies the name of the pCell definition object that you want to delete.

Return

MWXSuccess	Occurs if successfully delete the pCell definition.
MWXFail	Occurs if fail to delete the pCell definition.

Example

```
rc = MWXDb_Delete_PcellDefinition(libId, pcellObjName);
switch (rc) {
    case MWXSuccess:
        return;
    case MWXFail:
        printf("Fail to delete the Pcell definition object\n");
        return 0;
    Default:
        printf("Unknown return code \n");
        return 0;
}
```

MWXDb_Delete_ScanChain

Deletes a scan chain from the open cell.

Syntax

```
int MWXDb_Delete_ScanChain (
    MWXCellId_t cellId, /* INPUT */
    char scanChainName /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that owns the scan chain.
scanChainName	Specifies the name of the scan chain to be deleted.

Return

MWXNoLicense	Occurs if no software license is available.
MWXSuccess	Occurs if the scan chain is successfully deleted.
MWXFail	Occurs if the scan chain cannot be successfully deleted.

Example

```
rc = MWXDb_Delete_ScanChain( cellId, scanChainName );
switch (rc) {
    case MWXSuccess:
        return (MWX_TRUE_SCM);
    case MWXFail:
        return (MWX_FALSE_SCM);
}
```

Scheme Equivalent

dbDeleteScanChain (see the *IC Layout Command Reference Manual*)

MWXDb_Dump_AllVarRouteRules

Writes all the variable-width routing rules in the specified cell to the specified file.

Note:

Shield width and shield spacing rules for a particular rule set will *not* be dumped unless a shield rule is also defined in that rule set.

Syntax

```
int MWXDb_Dump_AllVarRouteRules (
    MWXCellId_t cellId, /*INPUT*/
    char *fileName /*INPUT*/
)
```

Parameters

cellId	Specifies the database identifier of the cell in which you want to dump the variable-width routing rules.
fileName	Specifies the name of the file to write the information into.

Return

MWXSuccess	Occurs if the dump is successful.
MWXNoLibraryOpened	Occurs if there is no library open.
MWXInvalidCellId	Occurs if the specified cell is 0.
MWXFail	Occurs if the dump failed for any other reason.

Example

```
rc = MWXDb_Dump_AllVarRouteRules (cellId, fileName);

switch (rc) {
    case MWXSuccess:
        return (MWX_TRUE_SCM);
    case MWXFail:
        MWX_Printf ("ERROR -- Failed to dump variable-width routing rules!\n");
        return (MWX_FALSE_SCM);
    default:
        MWX_Printf("ERROR -- unknown return code from
                    MWXDb_Dump_AllVarRouteRules\n");
        return (MWX_FALSE_SCM);
}
```

Scheme Equivalent

dbDumpAllVarRouteRules

MWXDb_Force_UpdateCellHierarchy

Forces the system to update all cell instances of a specified cell and make them consistent.

Syntax

```
int MWXDb_Force_UpdateCellHierarchy (
    MWXCellId_t cellId /* INPUT */
)
```

Parameters

`cellId` The ID of any open cell in the library.

Return

`MWXSuccess` Occurs if the system is successfully forced to update the cell hierarchy.

`MWXFail` Occurs if the system is not successfully forced to update the cell hierarchy.

Example

```
if (MWXDb_Force_UpdateCellHierarchy(cellId)
    != MWXSuccess) {
    MWX_Printf("Fail to force update cellHierarchy\n");
    return (MWX_FALSE_SCM);
}

return (MWX_TRUE_SCM);
```

Scheme Equivalent

`dbForceUpdateHierarchy` (see the Milkyway Environment Extension Language Reference Manual)

MWXDb_Get_AllNetVarRouteRules

Gets the IDs of all nets that have been assigned a rule set (and their corresponding rule set) from the specified cell ID. It is your responsibility to free the `ruleNames` and `netIdArray` memory locations.

Syntax

```
int MWXDb_Get_AllNetVarRouteRules(
    MWXCellId_t cellId, /*INPUT*/
    int32 *numOfRules, /*OUTPUT*/
    MWXPropName_t **ruleNames, /*OUTPUT*/
    MWXObjId_t **netIdArray /*OUTPUT*/
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell from which to get the netId-ruleName pairs.
<code>numOfRules</code>	Returns the number of nets that have been assigned a rule set.
<code>ruleNames</code>	Returns an array of rule-set names that have been assigned to the corresponding index in the <code>netIdArray</code> . It is your responsibility to free the memory of the array.
<code>netIdArray</code>	Returns an array of object identifiers for the nets in the specified <code>cellId</code> that are assigned rule sets. It is your responsibility to free the memory of the array.

Return

<code>MWXSuccess</code>	Occurs if the IDs are successfully retrieved.
<code>MWXFail</code>	Occurs if the IDs are not retrieved for any other reason.

Example

```
rc = MWXDb_Get_AllNetVarRouteRules(cellId, &numOfRules, &ruleNames, &netIds);

if (rc == MWXSuccess) {
    for (i = 0; i < numOfRules; i++) {
        if (MWXDb_Get_Net_name(cellId, netIds[i], &netName) != MWXSuccess) {
            MWX_Printf("Failed to get Net name\n");
            exit(MWXFail);
        }
        MWX_Printf((char *) "netName %s, ruleName %s\n", netName, ruleNames[i]);
    }
} else {
    MWX_Printf("Failed to get varRoute rule names.\n");
    return (MWX_FALSE_SCM);
}

if (numOfRules) {
    free (ruleNames);
    free (netIds);
}

return (MWX_TRUE_SCM);
```

MWXdB_Get_AllRouteGuides

Get all the route guide database identifiers for the given cell. It is the user's responsibility to recover the memory allocated for the `objectIdArray` when the data are no longer needed.

Syntax

```
int MWXdB_Get_AllRouteGuides (
    MWXCellId_t    cellId,          /* INPUT */
    int32           *objectCount,    /* INPUT */
    MWXObjId_t     **objectIdArray  /* INPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell.
<code>objectCount</code>	Specifies the number of route guides returned in <code>objectIdArray</code> .
<code>objectIdArray</code>	Specifies the array of route guide database identifiers.

Return

<code>MWXSuccess</code>	Occurs if the function is successful.
<code>MWXFail</code>	Occurs if the function is unsuccessful.

Example

```
int get_AllRouteGuides_layerInfo ( MWXCellId cellId )
{
    int32 layer;
    int32 objCount;
    MWXObjId_t *routeGuideIdArray;
    if (MWXdB_Get_AllRouteGuides( cellId, &objCount, &routeGuideIdArray) !=
        MWXSuccess) {
        MWX_Printf("failed to get all routeGuide object IDs\n");
        return (MWXFail);
    }

    MWX_Printf("object count : %d\n", objCount);
    for (i=0; i<objCount; i++) {
        MWX_Printf("routeGuide object ID : %d\n", routeGuideIdArray[i]);
        if (MWXdB_Get_RouteGuide_layer(cellId, routeGuideIdArray[i],
            &layer) == MWXSuccess) {
            MWX_Printf("Layer number %d\n", layer);
        }
        else {
            MWX_Printf("Fail to get layer number of the routeGuide\n");
            return MWXFail;
        }
    }
}
```

```

    if (objCount) {
        free(routeGuideIdArray);
    }

    return (MWXSuccess);
}

```

MWXDb_Get_AllVarRouteRules

Gets the names of all variable-width routing rule sets defined in the specified cell.

Syntax

```

int MWXDb_Get_AllVarRouteRules(
    MWXCellId_t cellId,      /*INPUT*/
    int32 *numOfRules, /*OUTPUT*/
    MWXPropName_t **ruleNames /*OUTPUT*/
)

```

Parameters

cellId	Specifies the database identifier of the cell in which to retrieve the rule set names.
numOfRules	Returns the number of rule sets being returned in ruleNames.
ruleNames	Returns an array containing the names of all the rule sets defined in the specified cell.

Return

MWXSuccess	Occurs if the variable-width routing rules are retrieved successfully.
MWXNoLibraryOpened	Occurs if there is no library open.
MWXInvalidCellId	Occurs if the specified cell is 0.
MWXFail	Occurs if the variable-width routing rules are not retrieved successfully.

Example

```

rc = MWXDb_Get_AllVarRouteRules(cellId, &numOfRules, &ruleNames);
if (rc == MWXSuccess) {

    for (i = 0; i < numOfRules; i++)
        MWX_Printf("ruleName %s\n", ruleNames[i]);

} else {

```



```

MWX_Printf("Failed to get varRoute rule names.\n");
return (MWX_FALSE_SCM);
}

if (numOfRules) {
    free (ruleNames);
}

return (MWX_TRUE_SCM);

```

MWXDb_Get_Cell_EEQClass

Gets an array of CellClassNameList structure specifying electrical-equivalent class number for all cells in the open library. It is your responsibility to recover the memory allocated for nameListPtr when the data is no longer needed.

Syntax

```

int MWXDb_Get_Cell_EEQClass (
    int32 *numberOfCell, /* OUTPUT */
    MWXCellClassNameList_t *nameListPtr/* OUTPUT */
)

```

Parameters

<code>numberOfCell</code>	Specifies the number of cells in the library.
<code>nameListPtr</code>	Address of <code>CellClassNameList</code> structure array. It your responsibility to free the memory—for example, <code>free (nameListPtr)</code>

Return

<code>MWXNoLibraryOpened</code>	Occurs if library is not open.
<code>NoLicense</code>	Occurs if software license is not available.
<code>MWXFail</code>	Occurs if the <code>CellClassNameList</code> structure array is not successfully retrieved.
<code>MWXSuccess</code>	Occurs if the <code>CellClassNameList</code> structure array is successfully retrieved.

Example

```
rc = MWXDb_Get_Cell_EEQClass ( &numberOfCell, &nameListArray);

switch (rc) {
    case MWXSuccess:
        if (numberOfCell > 0) free (nameListName);
        return (MWXSc_Make_IntegerSCM (numberOfCell));
    case MWXNoLibraryOpened:
        return (MWXSc_Make_IntegerSCM (rc));
    case MWXFail:
        return (MWX_FALSE_SCM);
}
```

MWXDb_Get_Cell_LEQClass

Gets an array of `CellClassNameList` structure specifying logical-equivalent class number for all cells in the open library. It is your responsibility to recover the memory allocated for `nameListPtr` when the data is no longer needed.

Syntax

```
int MWXDb_Get_Cell_LEQClass (
    int32 *numberOfCell, /* OUTPUT */
    MWXCellClassNameList_t *nameListPtr /* OUTPUT */
)
```

Parameters

numberOfCell	Specifies the number of cells in library.
nameListPtr	The address of CellClassNameList structure array. It is your responsibility to free the memory—for example, <code>free (nameListPtr)</code>

Return

MWXNoLibraryOpened	Occurs if the library is not open.
NoLicense	Occurs if a software licence is not available.
MWXFail	Occurs if the CellClassNameList structure array is not successfully retrieved.
MWXSuccess	Occurs if the CellClassNameList structure array is successfully retrieved.

Example

```
rc = MWXDb_Get_Cell_LEQClass ( &numberOfCell, &nameListArray);

switch (rc) {
    case MWXSuccess:
        if (numberOfCell > 0) free (nameListName);
        return(MWXSc_Make_IntegerSCM (numberOfCell));
    case MWXNoLibraryOpened:
        return (MWXSc_Make_IntegerSCM (rc));
    case MWXFail:
        return (MWX_FALSE_SCM);
}
```

MWXDb_Get_CellInst_PowerDelayScale

Retrieves the DelayScale of the cellInstPower object from the database.

Syntax

```
int MWXDb_Get_CellInst_PowerDelayScale (
    MWXCellId_t    cellId,           /* INPUT */
    MWXObjId_t     objId,           /* INPUT */
    int32          *delayScale       /* OUTPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that owns the cellInstPower object.
objId	Specifies the database identifier of the cellInstPower object from which the delayScale will be retrieved.
delayScale	The delayScale value of the cellInstPower object.

Return

MWXSuccess	Occurs if the powerDelayScale is successfully retrieved.
MWXFail	Occurs if the powerDelayScale is not successfully retrieved.

Example

```
rc = MWXDb_Get_CellInst_PowerDelayScale(cellId, cellInstPowerId,
                                         &delayScale);
if (rc == MWXSuccess) {
    MWX_Printf("delayScale : %d\n", delayScale);
    return (MWX_TRUE_SCM);
}
else {
    MWX_Printf("MWXDb_Get_CellInst_PowerDelayScale fail\n");
    return (MWX_FALSE_SCM);
}
```

MWXDb_Get_CellInst_PowerId

Retrieves the power ID of the cellInstPower object.

Syntax

```
int MWXDb_Get_CellInst_PowerId(
    MWXCellId_t    cellId,          /* INPUT */
    MWXObjId_t     objId,          /* INPUT */
    MWXObjId_t     *nObjId         /* OUTPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that owns the cellInstPower object.
objId	Specifies the database identifier of the cellInst object from which the power ID will be retrieved.
nObjId	Specifies the database identifier of the cellInstPower object.

Return

MWXSuccess	Occurs if the powerId is successfully retrieved.
MWXFail	Occurs if the powerId is not successfully retrieved.

Example

```
MWX_Printf(" ***** Get cellInstPower record ***** \n");
        rc = MWXDb_Get_CellInst_PowerId (cellId, cellInstId,
                                          &cellInstPowerId);

switch (rc) {
    case MWXSuccess:
        MWX_Printf("Cell Instance Power Id : %d\n", cellInstPowerId);
        return (MWX_TRUE_SCM);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to get cell instance power id\n");
        return (MWX_FALSE_SCM);
    default:
        MWX_Printf("ERROR -- unknown return code \n");
        return (MWX_FALSE_SCM);}
```

MWXDb_Get_CellInst_PowerSupplyVoltage

Retrieves the supply voltage of the cellInstPower object from the database.

Syntax

```
int MWXDb_Get_CellInst_PowerSupplyVoltage (
    MWXCellId_t    cellId,          /* INPUT */
    MWXObjId_t     objId,          /* INPUT */
    uint32         *supplyVoltage  /* OUTPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that owns the cellInstPower object.
objId	Specifies the database identifier of the cellInstPower object from which the supply voltage will be retrieved.
supplyVoltage	The supply voltage value of the cellInstPower object

Return

MWXSuccess	Occurs if the supply voltage is successfully retrieved.
MWXFail	Occurs if the supply voltage is not successfully retrieved.

Example

```
rc = MWXDb_Get_CellInst_PowerSupplyVoltage(cellId, cellInstPowerId,
                                             &supplyVoltage);
if (rc == MWXSuccess) {
    MWX_Printf("supplyVoltage : %d\n", supplyVoltage);
    return (MWX_TRUE_SCM);
}
else {
    MWX_Printf("MWXDb_Get_CellInst_PowerSupplyVoltage fail\n");
    return (MWX_FALSE_SCM);
}
```

MWXDb_Get_CellInst_PowerTemperature

Retrieves temperature of the cellInstPower object.

Syntax

```
int MWXDb_Get_CellInst_PowerTemperature(
    MWXCellId_t    cellId,          /* INPUT */
    MWXObjId_t     objId,          /* INPUT */
    int32          *powerTemperature /* OUTPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that owns the cellInstPower object.
objId	Specifies the database identifier of the cellInstPower object from which the temperature will be retrieved.
powerTemperature	The temperature of the cellInstPower object.

Return

MWXSuccess	Occurs if the power temperature is successfully retrieved.
MWXFail	Occurs if the power temperature is not successfully retrieved.

Example

```
rc = MWXDb_Get_CellInst_PowerTemperature (cellId, cellInstId,
                                           &temperature);
switch (rc) {
    case MWXSuccess:
        MWX_Printf("temperature : %d\n", temperature);
        return(MWX_TRUE_SCM);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to get cell instance powerTemperature\n");
        return (MWX_FALSE_SCM);
    default:
        MWX_Printf("ERROR -- unknown return code \n");
        return (MWX_FALSE_SCM);
}
```

MWXDb_Get_CellInst_PowerValue

Retrieves the powerValue of the specified cellInstPower object.

Syntax

```
int MWXDb_Get_CellInst_PowerValue(
    MWXCellId_t    cellId,        /* INPUT */
    MWXObjId_t     objId,         /* INPUT */
    int32          *powerValue    /* OUTPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell that owns the <code>cellInstPower</code> object.
<code>objId</code>	Specifies the database identifier of the <code>cellInstPower</code> object from which the power value will be retrieved.
<code>powerValue</code>	The power value of the <code>cellInstPower</code> object.

Return

<code>MWXSuccess</code>	Occurs if the power value is successfully retrieved.
<code>MWXFail</code>	Occurs if the power value is not successfully retrieved.

Example

```
rc = MWXDb_Get_CellInst_PowerValue(cellId, cellInstPowerId,&powerValue);
    if (rc == MWXSuccess) {
        MWX_Printf("powerValue : %d\n", powerValue);
        return (MWX_TRUE_SCM)
    }
    else {
        MWX_Printf("MWXDb_Get_CellInst_PowerValue fail, code = %d\n", rc);
        return (MWX_FALSE_SCM);
    }
```

MWXDb_Get_CellRegionIdFromCellInstId

Retrieves the database identifier for the `cellInst` attached to a `cellRegion`. The output parameters of the function are undefined if the function does not return `MWXSuccess`.

Syntax

```
int MWXDb_Get_CellRegionIdFromCellInstId (
    MWXCellId_t cellId, /* INPUT: CellId */
    MWXObjId_t cellInstId, /* INPUT: Cell Inst Id */
    MWXObjId_t *cellRegionId /* OUTPUT: Cell Region Id */
)
```


Parameters

<code>cellId</code>	Specifies the database identifier of the cell containing the cell instance.
<code>cellInstId</code>	Specifies the database identifier of the cell instance whose <code>cellRegion</code> is being retrieved.
<code>cellRegionId</code>	Specifies the database identifier of the <code>cellRegion</code> being retrieved

Return

<code>MWXSuccess</code>	Occurs if the database identifier is successfully retrieved.
<code>MWXFail</code>	Occurs if the database identifier is not successfully retrieved.

Example

```
if (MWXDb_Get_CellRegionIdFromCellInstId(cellId, cellInstId, &cellRegionId) !=
MWXSuccess) {
    MWX_Printf("Fail to get CellRegionId from cellInstId \n");
    return (MWX_FALSE_SCM);
}
return (MWXSc_Make_DBIIdSCM(cellRegionId));
```

Scheme Equivalent

`dbGetRegionIdByCellInstId` (see the *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Get_DataTypeName

Gets the data type name, layer name, and data type number in the library.

Syntax

```
int MWXDb_Get_DataTypeName (
    MWXLibId_t libId, /* INPUT */
    char *layerName, /* INPUT */
    int dataTypeNum, /* INPUT */
    char **dataTypeName /* OUTPUT */
)
```

Parameters

libId	Specifies the database identifier of the library from which to get the data type name.
layerName	Specifies the layer name to which the data type belongs.
dataTypeNum	Specifies the data type number.
dataTypeName	Specifies the output data type name.

Return

MWXSuccess	Occurs if the data type name is successfully retrieved.
MWXFail	Occurs if the data type name is not retrieved.

Example

```
rc = MWXDb_Get_DataTypeName (libId, "M2R", 1, &dataTypeName);

switch (rc) {
    case MWXSuccess:
        MWX_Printf("The datatype name is %s\n", dataTypeName);
        return (MWX_TRUE_SCM);
    case MWXFail:
        MWX_Printf("\nERROR -- Failed to get datatype name\n");
        return (MWX_FALSE_SCM);
    break;
}

return MWX_TRUE_SCM;
```

Scheme Equivalent

dbGetDatatypeName

MWXDb_Get_GlobalNet_ByNameMap

Gets the global net ID by name map.

Syntax

```
int MWXDb_Get_GlobalNet_ByNameMap (
    MWXCellId_t cellId, /* INPUT */
    char *nameMap, /* INPUT */
    MWXObjId_t *netId /* OUTPUT */
)
```

Parameters

cellId	Specifies the database identifier number of the target cell.
nameMap	Specifies the name map of the global net.
netId	The database identifier number of the net.

Return

MWXSuccess	Occurs if successfully retrieved contact code index.
MWXFail	Occurs if fail to retrieve contact code index.

Example

```
rc = MWXDb_Get_GlobalNet_ByNameMap( cellId, nameMap, &netId);

switch (rc) {
    case MWXSuccess:
        return (netId);
    case MWXFail:
        printf("Fail to get the global net by name map\n");
        return 0;
    default:
        MWX_Printf("ERROR -- unknown return code \n");
        return 0;
}
```

MWXDb_Get_NetLoad

Retrieves the net load of a netPower object.

Syntax

```
int MWXDb_Get_NetLoad (
    MWXCellId_t cellId, /*INPUT*/
    MWXObjId_t netPowerId, /*INPUT*/
    float32 *load /*OUTPUT*/
)
```

Parameters

cellId	Specifies the database identifier of the cell that owns the netPower object.
netPowerId	Specifies the database identifier of the netPower from which the netLoad is going to be retrieved.
load	Specifies the net load of the netPower object.

Return

MWXSuccess	Occurs if the net load is successfully retrieved.
MWXFail	Occurs if the net load is not successfully retrieved.

Example

```
rc = MWXDb_Get_NetLoad(cellId, netPowerId, &load);
    if (rc == MWXSuccess) {
        MWX_Printf("load : %lf\n", load);
        return (MWX_TRUE_SCM);
    }
    else {
        MWX_Printf("MWXDb_Get_NetLoad fail, code = %d\n", rc);
        return (MWX_FALSE_SCM);
    }
```

MWXDb_Get_NetPowerId

Retrieves NetPower database identifiers.

Syntax

```
int MWXDb_Get_NetPowerId(
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t netId, /* INPUT */
    MWXObjId_t *netPowerId /* OUTPUT */
)
```

Parameters

cellId	Specifies the database ID of the cell that owns the netPower object.
netId	Specifies the database ID of the net from which to get the netPowerId.
netPowerId	Specifies the database ID of the netPower that is going to be retrieved.

Return

MWXSuccess	Occurs when the netPowerId is successfully retrieved.
MWXFail	Occurs when retrieval of the netPowerId fails.

Example

```
rc = MWXDb_Get_NetPowerId (cellId, netId, &netPowerId);
    switch (rc) {
        case MWXSuccess:
            MWX_Printf("Net Power Id : %d\n", netPowerId);
            return (MWX_TRUE_SCM);
        case MWXFail:
            MWX_Printf("ERROR -- Failed to get net power id\n");
            return (MWX_FALSE_SCM);
        default:
            MWX_Printf("ERROR -- unknown return code from
MWXDb_Get_NetPowerId\n");
            return (MWX_FALSE_SCM);}
```

MWXDb_Get_NetSwitchActivity

Retrieves the switch activity of the netPower object.

Syntax

```
int MWXDb_Get_NetSwitchActivity (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t netPowerId, /* INPUT */
    int32 *switchActivity /* OUTPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell that owns the <code>netPower</code> object.
<code>netPowerId</code>	Specifies the database identifier of the <code>netPower</code> from which the <code>netSwitchActivity</code> is going to be retrieved.
<code>switchActivity</code>	Specifies the <code>switchActivity</code> of the <code>netPower</code> object.

Return

<code>MWXSuccess</code>	Occurs if the <code>netSwitchActivity</code> is successfully retrieved.
<code>MWXFail</code>	Occurs if the <code>netSwitchActivity</code> is not successfully retrieved.

Example

```
rc = MWXDb_Get_NetSwitchActivity(cellId, netPowerId, &switchActivity);
    if (rc == MWXSuccess) {
        MWX_Printf("SwitchActivity : %d\n", switchActivity);
        return (MWX_TRU_SCM);
    }
    else {
        MWX_Printf("MWXDb_Get_NetSwitchActivity fail, code = %d\n", rc);
        return (MWX_FALSE_SCM);}
```

MWXDb_Get_RouteGuide

This command displays route guide information for a specified cell, route guide object, and field name.

Syntax

```
int MWXDb_Get_RouteGuide(
    MWXCellId_t cellId, /*INPUT*/
    MWXObjId_t routeGuideId, /*INPUT*/
    char *fieldName /*INPUT*/
```

Parameters

cellId	Cell identifier in the current library.
routeGuideId	Route guide identifier in the specified cell.
fieldName	Field name string for which you want information.

Return

MWXSuccess	Occurs when the route guide is successfully displayed.
MWXFail	Occurs when the route guide is not successfully displayed.

Example

```
rc = MWXDb_Get_RouteGuide(cellId, routeGuideId, fieldName);
switch (rc) {
    case MWXSuccess:
        MWX_Printf("\n***** GetRouteGuide is successful **** (rc=%d)\n",
            rc);
        return (MWX_TRUE_SCM);
    case MWXFail:
        MWX_Printf("\nERROR -- Failed to get routeguide\n");
        return (MWX_FALSE_SCM);
    case MWXNoLicense:
        MWX_Printf("\nERROR -- No License to run this function!\n");
        return (MWX_FALSE_SCM);
    default:
        MWX_Printf("\nERROR -- unknown return code from
MWXDb_GetRouteGuide\n");
        return (MWX_FALSE_SCM)
}
```

Scheme Equivalent

axQueryRouteGuide (see the *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Get_RouteGuide_PropertyValueArray

Gets all the property information of the specified route guides, including the count, names, data types, and values. The property data type should be of type integer. It is your responsibility to recover the memory allocated for the propNames, propDataType and propValueArray when the data are no longer needed.

Syntax

```
int MWXDb_Get_RouteGuide_PropertyValueArray (
    MWXCellId_t      cellId,          /* INPUT */
    MWXObjId_t       routeGuideId,    /* INPUT */
    ...)
```

MWXDb_Get_RouteGuide_PropertyValueArray

```

    int32          *propCount,          /* OUTPUT */
    MWXPropName_t  **propNames,         /* OUTPUT */
    unsigned char  **propDataType,      /* OUTPUT */
    MWXPropValue_t **propValueArray     /* OUTPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell.
routeGuideId	Specifies the database identifier of the routeGuide.
propCount	Specifies the property count.
propNames	Specifies the array of property name.
propDataType	Specifies the array of property data type.
propValueArray	Specifies the array of property value.

Return

MWXSuccess	Occurs if the function is successful.
MWXFail	Occurs if the function is unsuccessful.

Example

```

int getRouteGuideProp (
    MWXCellId_t cellId,
    MWXObjId_t routeGuideId) {

    int      rc;
    int32    propCount;
    MWXPropName_t  *propNames;
    unsigned char  *propDataType;
    MWXPropValue_t *propValueArray;

    rc = MWXDb_Get_RouteGuide_PropertyValueArray(cellId, objId, &propCount,
    &propName, &propDataType, &propValueArray);
    if (rc == MWXSuccess) {
        for (i = 0; i < propCount; i++) {
            MWX_Printf((char *)"prop name: %s\n", propName[i]);
            if (propDataType[i] == MWXcPropIntegerType) {
                MWX_Printf((char *)"data type: integer\n");
                MWX_Printf((char *)"prop value: %d\n", propValueArray[i].pInt);
            }
            else {
                MWX_Printf((char *)"Error: unknown data type: %d\n",
                propDataType[i]);
                return (MWXFail);
            }
        }
    }
}

```



```

    }
    else {
        MWX_Printf((char *) "Failed to get routeGuide property data\n");
        return (MWXFail);
    }

    if (propName) free(propName);
    if (propDataType) free(propDataType);
    if (propValueArray) free(propValueArray);

    return (MWXSuccess);
}

```

MWXDb_Get_TopHierCellInst

Gets the topmost hierarchical cell instance.

Syntax

```

int int MWXDb_Get_TopHierCellInst (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t *hierCellInstId /* OUTPUT */
)

```

Parameters

cellId	Specifies the database identifier of the cell that is opened.
hierCellInstId	Specifies the database identifier of the topmost hierarchical cell instance that is retrieved.

Return

MWXSuccess	Occurs if the command successfully gets the topmost hierarchical cell instance.
MWXFail	Occurs if the command does not get the topmost hierarchical cell instance.

Example

```

MWXObjId_t getTopHierCellInstance (
    MWXCellId_t cellId
)
{
    MWXObjId_t hierCellInstId;
    if (MWXDb_Get_TopHierCellInst(cellId,&hierCellInstId) != MWXSuccess) {
        MWX_Printf((char *) "Fail to get topHierCellInstId \n");
        return 0;
    }
    return hierCellInstId;
}

```

See Also

[MWXDb_Set_TopHierCellInst](#)

MWXDb_Get_VarRouteRuleName_ByNetId

Gets the name of the variable-width routing rule assigned to the given `netId`.

Syntax

```
int MWXDb_Get_VarRouteRuleName_ByNetId (
    MWXCellId_t cellId, /*INPUT*/
    MWXObjId_t netId,   /*INPUT*/
    char **ruleName /*OUTPUT*/
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell in which you want to fetch the variable-width routing rule name from the specified net.
<code>netId</code>	Specifies the database identifier of the net to get the rule name.
<code>ruleName</code>	Specifies the name of the rule set you want this rule to be defined in. If the rule set has not already been defined, this function will define it.

Return

<code>MWXNoLibraryOpened</code>	Occurs if there is no library currently open.
<code>MWXInvalidCellId</code>	Occurs if the cell Id is invalid.
<code>MWXNullStringValue</code>	Occurs if no <code>varRoute</code> rule is defined in the given net.
<code>MWXSuccess</code>	Occurs if the routing rule name is successfully retrieved.
<code>MWXFail</code>	Occurs if the routing rule could not be retrieved.

Example

```
rc = MWXDb_Get_VarRouteRuleName_ByNetId (cellId, netId, &str);

if (rc!=MWXSuccess) {
    MWX_Printf ("ERROR---Fail to get variable-width routing rule name!\n");
    return (MWX_FALSE_SCM);
}
```

```

}

if (str != NULL)
return (MWXSc_Make_StringSCM(str));
else
return (MWX_FALSE_SCM);

```

Scheme Equivalent

dbGetNetVarRouteRuleName

MWXDb_Get_VarRouteRuleSpacing_ByName

Gets all the spacing or shield-spacing rules from the specified rule set.

Syntax

```

int MWXDb_Get_VarRouteRuleSpacing_ByName (
    MWXCellId_t cellId, /*INPUT*/
    char *ruleName, /*INPUT*/
    int32 shieldOrNot, /*INPUT*/
    int32 *count, /*OUTPUT*/
    MWXVarRouteRule1_t **valueArray /*OUTPUT*/
)

```

Parameters

cellId	Specifies the database identifier of the cell for which you want to fetch the variable-width routing rules.
ruleName	Specifies the name of the rule set you want this rule to be defined in. If the rule set has not already been defined, this function will define it.
shieldOrNot	Specifies whether to get spacing rules or shield spacing rules. If <code>shieldOrNot</code> is set to zero, then spacing rules are retrieved.
count	Returns the number of spacing rules returned in <code>valueArray</code> .
valueArray	Returns an array of all spacing or shield-spacing rules from <code>ruleName</code> , depending on the value of <code>shieldOrNot</code> . It is your responsibility to free the memory of the array.

Return

MWXSuccess	Occurs if the rule is successfully retrieved.
MWXNoLibraryOpened	Occurs if there is no library open.
MWXInvalidCellId	Occurs if the specified cell is 0.
MWXFail	Occurs if the rule could not be fetched for any other reason.

Example

```
MWXSCM_t varRoute[MWXcMaxVarRouteRulePerSet];
MWXVarRouteRule1_t *valueArray=NULL;
MWXSCM_t rtnRoute, route[4];
int32 flag=0;

rc = MWXDb_Get_VarRouteRuleSpacing_ByName (cellId, ruleName,
                                           flag, &count, &valueArray);

if (rc != MWXSuccess)
return (MWX_FALSE_SCM);

route[0] = MWXSc_Make_StringSCM("spacing");
for (i=0; i<count; i++) {
sprintf (str, "\"%s\"", valueArray[i].layerName);
route[1] = MWXSc_Make_StringSCM(str);
route[2] = MWXSc_Make_DoubleSCM(valueArray[i].value);
varRoute[i] = MWXSc_Make_ListSCM (3, route);
}
```

```
free (valueArray);  
rtnRoute = MWXSc_Make_ListSCM (i, varRoute);  
return rtnRoute;
```

Scheme Equivalent

dbGetVarRouteRule

MWXDb_Get_VarRouteRuleTaperLevel_ByName

Gets the taper-level or shield rule from the specified rule set.

Syntax

```
int MWXDb_Get_VarRouteRuleTaperLevel_ByName (  
    MWXCellId_t cellId, /*INPUT*/  
    char *ruleName, /*INPUT*/  
    int32 shieldOrNot, /*INPUT*/  
    int32 *count, /*OUTPUT*/  
    int32 **valueArray /*OUTPUT*/  
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell for which you want to get the variable-width routing rules.
<code>ruleName</code>	Specifies the name of the rule set you want this rule to be defined in. If the rule set has not already been defined, this function will define it.
<code>shieldOrNot</code>	Specifies whether to get the taper-level or shield rule. If <code>shieldOrNot</code> is set to zero, then the taper level rule is retrieved.
<code>count</code>	Returns the number of spacing rules returned in <code>valueArray</code> .
<code>valueArray</code>	Returns an array containing the taper-level rule or shield rule from <code>ruleName</code> , depending on the value of <code>shieldOrNot</code> . It is your responsibility to free the memory of the array.

Return

<code>MWXSuccess</code>	Occurs if the rule is successfully retrieved.
<code>MWXNoLibraryOpened</code>	Occurs if there is no library open.
<code>MWXInvalidCellId</code>	Occurs if the specified cell is 0.
<code>MWXFail</code>	Occurs if the rule could not be retrieved for any other reason.

Example

```
MWXSCM_t varRoute[MWXcMaxVarRouteRulePerSet];
MWXSCM_t rtnRoute, route[4];
int32 flag=0;
int32 *valueArray=NULL;

rc = MWXDb_Get_VarRouteRuleTaperLevel_ByName (cellId, ruleName,
                                              flag, &count, &valueArray);

if (rc != MWXSuccess)
    return (MWX_FALSE_SCM);

route[0] = MWXSc_Make_StringSCM("tapeLevel");
for (i=0; i<count; i++) {
    route[1] = MWXSc_Make_IntegerSCM(valueArray[i]);
    varRoute[i] = MWXSc_Make_ListSCM (2, route);
}

free (valueArray);
```

```
rtnRoute = MWXSc_Make_ListSCM (i, varRoute);  
return rtnRoute;
```

Scheme Equivalent

dbGetVarRouteRule

MWXDb_Get_VarRouteRuleWidth_ByName

Gets all the width or shield-width rules from the specified rule set.

Syntax

```
int MWXDb_Get_VarRouteRuleWidth_ByName (  
    MWXCellId_t cellId, /*INPUT*/  
    char *ruleName, /*INPUT*/  
    int32 shieldOrNot, /*INPUT*/  
    int32 *count, /*OUTPUT*/  
    MWXVarRouteRule1_t **valueArray /*OUTPUT*/  
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell for which you want to get the variable-width routing rules.
<code>ruleName</code>	Specifies the name of the rule set you want this rule to be defined in. If the rule set has not already been defined, this function will define it.
<code>shieldOrNot</code>	Specifies whether to get width rules or shield width rules. If <code>shieldOrNot</code> is set to zero, then width rules are retrieved.
<code>count</code>	Returns the number of width rules returned in <code>valueArray</code> .
<code>valueArray</code>	Returns an array of all width or shield-width rules from <code>ruleName</code> , depending on the value of <code>shieldOrNot</code> . It is your responsibility to free the memory of the array.

Return

<code>MWXSuccess</code>	Occurs if the rule is successfully retrieved.
<code>MWXNoLibraryOpened</code>	Occurs if there is no library open.
<code>MWXInvalidCellId</code>	Occurs if the specified cell is 0.
<code>MWXFail</code>	Occurs if the rule could not be retrieved for any other reason.

Example

```
MWXSCM_t varRoute[MWXCMaxVarRouteRulePerSet];
MWXVarRouteRule1_t *valueArray=NULL;
MWXSCM_t rtnRoute, route[4];
int32 flag=0;

rc = MWXDb_Get_VarRouteRuleWidth_ByName (cellId, ruleName,
                                         flag, &count, &valueArray);

if (rc != MWXSuccess)
return (MWX_FALSE_SCM);

route[0] = MWXSc_Make_StringSCM("width");
for (i=0; i<count; i++) {
sprintf (str, "%s\\", valueArray[i].layerName);
route[1] = MWXSc_Make_StringSCM(str);
route[2] = MWXSc_Make_DoubleSCM(valueArray[i].value);
varRoute[j++] = MWXSc_Make_ListSCM (3, route);
}
```



```

free (valueArray);
rtnRoute = MWXSc_Make_ListSCM (j, varRoute);
return rtnRoute;

```

Scheme Equivalent

dbGetVarRouteRule

MWXDb_Get_VarVarRouteRuleContact_ByName

Gets all the contact rules from the specified rule set.

Syntax

```

int MWXDb_Get_VarVarRouteRuleContact_ByName (
    MWXCellId_t cellId, /*INPUT*/
    char *ruleName, /*INPUT*/
    int32 *count, /*OUTPUT*/
    MWXVarRouteRule2_t **valueArray /*OUTPUT*/
)

```

Parameters

cellId	Specifies the database identifier of the cell for which you want to define the variable-width routing rules.
ruleName	Specifies the name of the rule set you want this rule to be defined in. If the rule set has not already been defined, this function will define it.
count	Returns the number of contact rules returned in valueArray.
valueArray	Returns an array of all contact rules from ruleName. It is your responsibility to free the memory of the array.

Return

MWXSuccess	Occurs if the rule is successfully retrieved.
MWXNoLibraryOpened	Occurs if there is no library open.
MWXInvalidCellId	Occurs if the specified cell is 0.
MWXFail	Occurs if the rule could not be retrieved for any other reason.

Example

```

MWXSCM_t varRoute[MWXcMaxVarRouteRulePerSet];

```

MWXDb_Get_VarVarRouteRuleContact_ByName

```

MWXVarRouteRule2_t *valueArray=NULL;
MWXSCM_t rtnRoute, route[4];

rc = MWXDb_Get_VarRouteRuleContact_ByName (cellId, ruleName,
                                             &count, &valueArray);

if (rc != MWXSuccess)
    return (MWX_FALSE_SCM);

route[0] = MWXSc_Make_StringSCM("contact");
for (i=0; i<count; i++) {
    sprintf (str, "\\%s\\", valueArray2[i].contactName);
    route[1] = MWXSc_Make_StringSCM(str);
    route[2] = MWXSc_Make_IntegerSCM(valueArray[i].xtimes);
    route[3] = MWXSc_Make_IntegerSCM(valueArray[i].ytimes);
    varRoute[i] = MWXSc_Make_ListSCM (4, route);
}

rtnRoute = MWXSc_Make_ListSCM (i, varRoute);
return rtnRoute;

```

Scheme Equivalent

dbGetVarRouteRule

MWXDb_Iter_Next_ExpPinInst

Retrieves the next expanded pin instance by using the specified iterator ID. The pin instance data is generated during retrieval. Therefore, no object ID is associated with it.

Syntax

```

int MWXDb_Iter_Next_ExpPinInst(
    MWXCellId_t cellId, /* INPUT */
    MWXIteratorId_t expPinInstIterId /* INPUT */      MWXExpPinInst_t
    **expPinInstId /* OUTPUT */
)

```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell in which the expanded pin instance is to be iterated.
<code>expPinInstIterId</code>	Specifies the iterator identifier.
<code>ExpPinInstId</code>	Specifies the data record of the expanded pin instance.

Return

<code>MWXSuccess</code>	Occurs if the pin instance was successfully returned.
<code>MWXIterDone</code>	Occurs if the end of the pin instance iteration is reached.
<code>MWXFail</code>	Occurs if the pin instance was not successfully returned.

Dynamic Example

See the dynamic example in [MWXDb_Iter_PortInstToExpPinInst](#).

Scheme Equivalent

`dbFetchPinInstFromPortInst`

See Also

[MWXDb_Iter_PortInstToExpPinInst](#)
[MWXDb_Iter_Quit](#)

MWXDb_Iter_PortInstToExpPinInst

Gets the expanded pin instance ID (ExpPinInst iterator), which is used for iterating through the ExpPinInst attached file of a specified port instance (`PortInst`) from the specified cell.

Syntax

```
int MWXDb_Iter_PortInstToExpPinInst(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t portInstId, /* INPUT */  
    MWXIteratorId_t *ExpPinInstIterId /* OUTPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell in which the expanded pin instance is to be iterated.
portInstId	Specifies the database identifier of the port instance.
ExpPinInstIterId	Specifies the iterator identifier.

Return

MWXSuccess	Occurs if the iterator ID is successfully returned.
MWXFail	Occurs if Milkyway fails to iterate the port instance to the expanded pin instance.

Dynamic Example

```
#include<stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string.h>
#include "mwX.h"
#include "mwX_proto.h"

#ifdef __cplusplus
extern "C" {
    MWXSCM_t getPinInstFromPortInst();
    void reg_getPinInstFromPortInst();
}
#endif

MWXSCM_t getPinInstFromPortInst (MWXSCM_t list)
{
    char progDescrString[50] = "MW-CAPI Example
getPinInstFromPortInst -";
    static int caseSensitive = 0;
    char *name;
    MWXLibId_t libId;
    MWXCellId_t cellId;
    int rc;
    int i,j;
    MWXObjId_t portInstId;
    MWXIteratorId_t pinInstMasterIterId;
    MWXExpPinInst_t *expPinInst;
    int32 lengthPrecision, minimumResolution;
    char * lengthUnit;

    /*-----
    HELLO
    -----*/
```

```

MWX_Printf((char*)"\\n%s ...\\n", progDescrString);
if (MWXSc_Scanf (list, "ooo", &libId,&cellId,&portInstId)
    != 3) {
    MWX_Printf((char*)"Error.Synopsis : helloworld-api libId\\
cellId portInstId\\n");
    return (MWX_FALSE_SCM);
}

MWX_Printf("libId is %d, cellId is %d, portInstId is %d\\n",
    libId, cellId, portInstId);
rc = MWXDb_Get_TechResolution( &lengthPrecision,
&minimumResolution, &lengthUnit);
if (rc == MWXFail) {
    MWX_Printf((char *)"Error - Fail to get
TechResolution\\n");
    return (MWX_FALSE_SCM);
}
rc = MWXDb_Iter_PortInstToExpPinInst(cellId, portInstId,
&pinInstMasterIterId);
if (rc == MWXFail) {
    MWX_Printf((char *)"Error - Fail to iterate Expanded
PinInst from PortInst\\n");
    return (MWX_FALSE_SCM);
}
while ((rc = MWXDb_Iter_Next_ExpPinInst(cellId,
pinInstMasterIterId, &expPinInst)) == MWXSuccess) {
    MWX_Printf((char *)"PinInstMasterId :%d\\n",
        expPinInst->pinInstMasterId);
    MWX_Printf((char *)"NetId :%d\\n",expPinInst->netId);
    MWX_Printf((char *)"PinInst Name :%s\\n",expPinInst-
>name);
    MWX_Printf((char *)"Status Flag :%d\\n",
        expPinInst->statusFlag);
    MWX_Printf((char *)"LocalStatus :%d\\n",
        expPinInst->localStatus);
    MWX_Printf((char *)"Access direction :%d\\n",
        expPinInst->accessDirection);
    MWX_Printf((char *)"Status Flag1 :%d\\n",
        expPinInst->statusFlag1);
    MWX_Printf((char *)"CellInstId :%d\\n",
        expPinInst->cellInstId);
    MWX_Printf((char *)"PortInstId :%d\\n",
        expPinInst->portInstId);
    MWX_Printf((char *)"ChildPinId :%d\\n",
        expPinInst->childPinId);
    MWX_Printf((char *)"ChildNetId :%d\\n",
        expPinInst->childNetId);
    MWX_Printf((char *)"ChildPortId :%d\\n",
        expPinInst->childPortId);
    MWX_Printf((char *)"Electrical Equivalent Class :
%d\\n",
        expPinInst->electricalEquivalentClass);

```

```

    MWX_Printf((char *) "Must Join Class :%d\n",
                expPinInst->mustJoinClass);
    MWX_Printf((char *) "Layer :%d\n", expPinInst->layer);
    MWX_Printf((char *) "DataType :%d\n", expPinInst->
>dataType);
    MWX_Printf((char *) "BBox :(%7.3f %7.3f) (%7.3f
%7.3f)\n",
                1.0*expPinInst->bbox.left/lengthPrecision,
                1.0*expPinInst->bbox.bottom/lengthPrecision,
                1.0*expPinInst->bbox.right/lengthPrecision,
                1.0*expPinInst->bbox.top/lengthPrecision);
    MWX_Printf((char *) "NumberOfPoint :%d\n",
                expPinInst->numberOfPoint);
    MWX_Printf((char *) "Points :");
    for (i = 0; i < (abs(expPinInst->numberOfPoint)); i++) {
        MWX_Printf((char *) "(%7.3f %7.3f)",
                    1.0*expPinInst->points[i].x/lengthPrecision,
                    1.0*expPinInst->points[i].y/lengthPrecision);
    }
    MWX_Printf((char *) "\n");
}
if (rc == MWXIterDone) {
    MWXDb_Iter_Quit(pinInstMasterIterId);
}
else {
    MWX_Printf((char *) "Fail to query PinInst from
PortInst\n");
    return(MWX_FALSE_SCM);
}

/*-----
DONE
-----*/

MWX_Printf((char*) "%s GOODBYE! \n\n", progDescrString);
return(MWX_TRUE_SCM);
}
void reg_getPinInstFromPortInst(){
    MWXSc_Register_Function((char*) "getPinInstFromPortInst-
api", getPinInstFromPortInst);
}

```

Scheme Equivalent

dbFetchPinInstFromPortInst

See Also

[MWXDb_Iter_Next_ExpPinInst](#)

[MWXDb_Iter_Quit](#)

MWXdB_Iter_PortToTiming

Gets the timing iterator ID used for iterating through the timing attached file of a specified port from a specified cell.

Syntax

```
int MWXdB_Iter_PortToTiming(  
    MWXCellId_t cellId, /* INPUT */  
    MWXObjId_t portId, /* INPUT */  
    MWXIteratorId_t *timingIterId /* OUTPUT */  
)
```

Parameters

cellId	Specifies the database identifier for the specified cell.
portId	Specifies the database identifier of the port object from which you want to retrieve the timing iterator ID.
timingIterId	Specifies the returned timing iterator ID from a specified port.

Return

MWXSucces	Occurs if the timing iterator ID is successfully returned.
MWXIterDone	Occurs if the iterator has finished.
MWXFail	Occurs if the iterator fails.

Dynamic Example

```
#include<stdio.h>  
#include <stdlib.h>  
#include <malloc.h>  
#include <string.h>  
#include "mwX.h"  
#include "mwX_proto.h"  
  
#ifdef __cplusplus  
extern "C" {  
    MWXSCM_t getPortCap();  
    void reg_getPortCap();  
}  
#endif  
  
MWXSCM_t getPortCap (MWXSCM_t list)  
{  
    char progDescrString[50] = "MW-CAPI Example getPortCap -";  
    static int caseSensitive = 0;  
    char portName[256];
```

```

MWXLibId_t libId;
MWXCellId_t cellId;
int rc;
int i,j;
MWXObjId_t portId,portCapId;
MWXIteratorId_t timingIterId;
MWXExpPinInst_t *expPinInst;
MWXObjTypeId_t portCapType;
int32 riseMin,riseNom, riseMax, fallMin, fallNom, fallMax;

/*-----
HELLO
-----*/
MWX_Printf((char*)"\\n%s ...\\n", progDescrString);
if (MWXSc_Scanf (list, "os", &cellId,portName) != 2) {
    MWX_Printf((char*)"Error.Synopsis : getPortCap-api cellId
portName\\n");
    return (MWX_FALSE_SCM);
}

MWX_Printf("cellId is %d, portName is %s\\n",
    cellId, portName);
rc = MWXDb_Get_Port_ByName( cellId,portName, &portId);
if (rc == MWXFail) {
    MWX_Printf((char*)"Error - Fail to get portName\\n");
    return (MWX_FALSE_SCM);
}
rc = MWXDb_Iter_PortToTiming(cellId, portId,
&timingIterId);
if (rc == MWXFail) {
    MWX_Printf((char*)"Error - Fail to iterate PortToTiming
\\n");
    return (MWX_FALSE_SCM);
}
while ((rc = MWXDb_Iter_Next(timingIterId, &portCapId))
== MWXSuccess) {
    MWX_Printf((char*)"PortCapId :%d\\n",portCapId);
    if (MWXDb_Get_ObjectsType(cellId,portCapId,&portCapType)
!= MWXSuccess) {
        MWX_Printf((char*)"Fail to get ObjectsType\\n");
        return (MWX_FALSE_SCM);
    }
    if (portCapType == MWX_PORTCAP) {
        break;
    }
}
if ((portCapType == MWX_PORTCAP) || (rc == MWXIterDone)) {
    MWXDb_Iter_Quit(timingIterId);
}
else {
    MWX_Printf((char*)"Fail to query PinInst from
PortInst\\n");
    return(MWX_FALSE_SCM);
}

```



```

    }
    if (portCapType != MWX_PORTCAP) {
        MWX_Printf((char *) "There is no Port Capacitance
information for this port\n");
        return(MWX_FALSE_SCM);
    }
    if (MWXDb_Get_PortCap_capacitance(cellId, portCapId,
&riseMin, &riseNom, &riseMax,&fallMin,&fallNom,&fallMax) !=
MWXSuccess) {
        MWX_Printf((char*) "Error - Fail to get PortCap
capacitance\n");
        return(MWX_FALSE_SCM);
    }
    else {
        MWX_Printf((char*) "Rise Min = %d\n",riseMin);
        MWX_Printf((char*) "Rise Nom = %d\n",riseNom);
        MWX_Printf((char*) "Rise Max = %d\n",riseMax);
        MWX_Printf((char*) "Fall Min = %d\n",fallMin);
        MWX_Printf((char*) "Fall Nom = %d\n",fallNom);
        MWX_Printf((char*) "Fall Max = %d\n",fallMax);
    }

    /*-----
    DONE
    -----*/

    MWX_Printf((char*) "%s GOODBYE! \n\n",progDescrString);
    return(MWX_TRUE_SCM);
}
void reg_getPortCap(){
    MWXSc_Register_Function((char*) "getPortCap-
api",getPortCap);
}

```

MWXDb_LoadDesignSetupDone

Load the design setup file after running floorplan. You should open one cell before calling this function.

Syntax

```
int MWXDb_LoadDesignSetupDone (void)
```

Parameters

None

Return

MWXSuccess	Occurs after successful loading of the design setup file.
MWXFail	Occurs after failure to load the design setup file.
MWXNoLicense	Occurs when no license is available.
MWXNoLibraryOpened	Occurs when no library is open.
MWXNoCellOpened	Occurs when no cell is open.

Example

```
rc = MWXDb_LoadDesignSetupDone();
switch (rc) {
    case MWXSuccess:
        MWX_Printf(" Load Design Setup file succefully!\n");
        return (MWX_TRUE_SCM);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to Load Design Setup file!\n");
        return (MWX_FALSE_SCM);
    case MWXNoLicense:
        MWX_Printf("ERROR -- No License to Load Design Setup file!\n");
        return (MWX_FALSE_SCM);
    case MWXNoLibraryOpened:
        MWX_Printf("ERROR -- No Library Opened!\n");
        return (MWX_FALSE_SCM);
    case MWXNoCellOpened:
        MWX_Printf("ERROR -- No Cell Opened!\n");
        return (MWX_FALSE_SCM);
    default:
        MWX_Printf("ERROR -- Unknown error while Loading Design Setup
file!\n");
        return (MWX_FALSE_SCM);
}
```

Scheme Equivalent

axLoadDesignSetupDone (see the *Milkyway Environment Extension Language Reference Manual*)

Licenses Required

Apollo, Jupiter

MWXDb_Remove_InstGroup_FromScanChain

Deletes a scan cell instance group from the scan chain in the open cell.

Syntax

```
int MWXDb_Remove_InstGroup_FromScanChain (
    MWXCellId_t cellId, /* INPUT */
    char *scanChainName /* INPUT */
    char *instGroupName /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that owns the scan chain, group instance, and cell instances.
scanChainName	Specifies the scan chain name.
instGroupName	Specifies the name of the instance group.

Return

MWXFail	Occurs if the specified group cannot be successfully deleted from the scan chain.
MWXSuccess	Occurs if the specified group is successfully deleted.

Example

```
rc = MWXDb_Remove_InstGroup_FromScanChain (cellId, scanChainName,
instGroupName);
switch (rc){
    case MWXSuccess:
        return (MWX_TRUE_SCM)
    case MWXFail:
        MWX_Printf("ERROR -- function failed to remove the group\n");
        return (MWX_FALSE_SCM);
    default:
        MWX_Printf("ERROR -- unknown return code from
MWXDb_Remove_InstGroup_FromScanChain\n");
        return (MWX_FALSE_SCM);
}
```

Scheme Equivalent

dbRemoveInstGroupFromScanChain (see the *Milkyway Environment Extension Language Reference Manual*)

MWXdB_Reset_CellInstance_Traceable

Resets traceability for all cell instances in a cell.

Syntax

```
int MWXdB_Reset_CellInstance_Traceable (  
    MWXCellId_t cellId,    /* INPUT */  
)
```

Parameters

cellId	Specifies the cell to reset tractability.

Return

MWXSuccess	Occurs if all cell instances in a cell are reset successfully as traceable.

MWXFail	Occurs if cell instances were not successfully reset as traceable.

Example

```
rc = MWXdB_Reset_CellInstance_Traceable(cellId);  
switch (rc) {  
    case MWXSuccess:  
        return (MWX_TRUE_SCM);  
    case MWXFail:  
        MWX_Printf("ERROR -- Failed to reset all cell instances traceable\n");  
        return (MWX_FALSE_SCM);  
    default:  
        MWX_Printf("ERROR -- unknown return code from  
MWXdB_Reset_CellInstance_Traceable\n");  
        return (MWX_FALSE_SCM);  
}
```

MWXdB_Reset_Net_Traceable

Reset traceability for all nets in a cell.

Syntax

```
int MWXdB_Reset_Net_Traceable (  
    MWXCellId_t cellId,    /* INPUT */  
)
```

Parameters

`cellId` Specifies the cell that contains the cell instance.

Return

`MWXSuccess` Occurs if the command successfully resets all nets in a cell to be traceable.

`MWXFail` Occurs if the command cannot reset all nets to be traceable.

Example

```
rc = MWXDb_Reset_Net_Traceable(cellId);
switch (rc) {
    case MWXSuccess:
        return (MWX_TRUE_SCM);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to reset all nets traceable\n");
        return (MWX_FALSE_SCM);
    default:
        MWX_Printf("ERROR -- unknown return code from MWXDb_Reset_Net_Traceable\n");
        return (MWX_FALSE_SCM);
}
```

MWXDb_Restore_HierDesign

Restores the current hierarchy design.

Syntax

```
int32 MWXDb_Archive_HierDesign(
    char *cellName, /*INPUT*/
    char *arLibName /*INPUT */
)
```

Parameters

cellName	Contains the cell name of the cell to archive. It can also specify which version of the cell to archive (optional.) It may also specify a new name for the cell in the archive library (optional.) Syntax: cellName[;versionNumber] [=newCellName] Example: "cellA=cellB" copies "cellA" from the source library to "cellB" in the destination library.
arLibName	Specifies the name of the archive library. If it already exists, only modified data is archived to the archived library. If the library does not exist, it is created.

Return

MWXSuccess	Occurs if Milkyway successfully the archives the hierarchy design.
MWXFail	Occurs if Milkyway fails to archive the hierarchy design.

Example

```
char * cellName = "toprt";
char * arLibName = "design2";
rc = MWXDb_Restore_HierDesign(cellName, arLibName);
switch (rc) {
    case MWXSuccess:
        return (MWX_TRUE_SCM);
    case MWXFail:
        MWX_Printf("ERROR -- failed to restore the hierarchy design \n");
        return (MWX_FALSE_SCM);
}
```

See Also

MWXDb_Archive_HierDesign

MWXDb_Set_CellInst_PowerDelayScale

Sets the delay scale of the cellInstPower in the database.

Syntax

```
int MWXDb_Set_CellInst_PowerDelayScale (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t objId, /* INPUT */
    int32 delayScale /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that owns the cellInstPower object.
objId	Specifies the database identifier of the cellInstPower for which the delay scale is going to be set.
delayScale	Specifies the delay scale of the cellInstPower object.

Return

MWXSuccess	Occurs when the delay scale is successfully set.
MWXFail	Occurs when the delay scale is not successfully set.

Example

```
rc = MWXDb_Set_CellInst_PowerDelayScale(cellId, cellInstPowerId,
    delayScale);
if (rc == MWXSuccess) {
    MWX_Printf("delayScale : %d\n", delayScale);
    return (MWX_TRUE_SCM);
}
else {
    MWX_Printf("MWXDb_Set_CellInst_PowerDelayScale fail\n");
    return (MWX_FALSE_SCM);
}
```

MWXDb_Set_CellInst_PowerSupplyVoltage

Sets the supply voltage of the cellInstPower in the database.

Syntax

```
int MWXDb_Set_CellInst_PowerSupplyVoltage (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t objId, /* INPUT */
    int32 supplyVoltage /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that owns the cellInstPower object.
objId	Specifies the database identifier of the cellInstPower for which the supply voltage is going to be set.
supplyVoltage	Specifies the supply voltage of the cellInstPower object.

Return

MWXSuccess	Occurs when the supply voltage is successfully set.
MWXFail	Occurs when the supply voltage is not successfully set.

Example

```
rc = MWXDb_Set_CellInst_PowerSupplyVoltage(cellId, cellInstPowerId,
      supplyVoltage);
if (rc == MWXSuccess) {
    MWX_Printf("supplyVoltage : %d\n", supplyVoltage);
    return (MWX_TRUE_SCM);
}
else {
    MWX_Printf("MWXDb_Set_CellInst_PowerSupplyVoltage fail\n");
    return (MWX_FALSE_SCM);}
```

MWXDb_Set_CellInst_PowerTemperature

Sets the temperature of the cellInstPower in the database.

Syntax

```
int MWXDb_Set_CellInst_PowerTemperature (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t objId, /* INPUT */
    int32 temperature /* INPUT */
)
```


Parameters

cellId	Specifies the database identifier of the cell that owns the cellInstPower object.
objId	Specifies the database identifier of the cellInstPower for which the temperature is going to be set.
temperature	Specifies the temperature of the cellInstPower object.

Return

MWXSuccess	Occurs when the temperature is successfully set.
MWXFail	Occurs when the temperature is not successfully set.

Example

```
rc = MWXDb_Set_CellInst_PowerTemperature(cellId, cellInstPowerId,
    temperature);
if (rc == MWXSuccess) {
    MWX_Printf("temperature : %d\n", temperature);
    return (MWX_TRUE_SCM);
}
else {
    MWX_Printf("MWXDb_Set_CellInst_PowerTemperature fail\n");
    return (MWX_FALSE_SCM);
}
```

MWXDb_Set_CellInst_PowerValue

Sets the power value of the cellInstPower in the database.

Syntax

```
int MWXDb_Set_CellInst_PowerValue (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t objId, /* INPUT */
    int32 powerValue /* INPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell that owns the <code>cellInstPower</code> object.
<code>objId</code>	Specifies the database identifier of the <code>cellInstPower</code> for which the power value is going to be set.
<code>powerValue</code>	Specifies the power value of the <code>cellInstPower</code> object.

Return

<code>MWXSuccess</code>	Occurs if the power value is successfully set.
<code>MWXFail</code>	Occurs if the power value is not successfully set.

Example

```
rc = MWXDb_Set_CellInst_PowerValue(cellId, cellInstPowerId, powerValue);
    if (rc == MWXSuccess) {
        MWX_Printf("powerValue : %d\n", powerValue);
        return (MWX_TRUE_SCM);
    }
    else {
        MWX_Printf("MWXDb_Set_CellInst_PowerValue fail\n");
        return (MWX_FALSE_SCM);
    }
```

MWXDb_Set_CellInstanceECOStatus_ByName

Sets the ECO status bits in the cell instance `statusFlag` field.

- `ECOReset` resets all ECO 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 `MWXDb_Create_CellInstance`, `dbCreateCellInst`, `geAddCell`, and `TDF insertPad` to indicate that this cell was interactively added by you and should not be deleted even if it does not appear in the ECO netlist.

Syntax

```
int MWXDb_Set_CellInstanceECOStatus_ByName(  
    MWXCellId_t cellId, /* INPUT */  
    char *cellInstanceName, /* INPUT */  
    char *ecoStatus /* INPUT */  
)
```

Parameters

cellId	Specifies the database identifier of the cell containing the cell instance being processed.
cellInstanceName	Specifies the instance name of the cell whose status is being set.
ecoStatus	Specifies the new status of the cell. Valid values are ECOReset, ECODelete, ECOAdd, ECOReplace, and ECOIgnore.

Return

MWXSuccess	Occurs if the status is successfully set.
MWXFail	Occurs if the status is not successfully set.

Example

```
rc = MWXDb_Set_CellInstanceECOStatus_ByName (cellId,  
    cellInstanceName, &ecoStatus);  
switch (rc) {  
    case MWXSuccess:  
        break;  
    case MWXFail:  
        MWX_Printf("ERROR -- Failed to set ECO status\n");  
        break;  
}
```

Scheme Equivalent

dbSetCellInstECOStatusByName (see the *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Set_CellInstance_Traceable

Sets a cell instance to be traceable or not traceable.

Syntax

```
int MWXDb_Set_CellInstance_Traceable (
    MWXCellId_t cellId,    /* INPUT */
    MWXObjId_t  ciId,      /* INPUT */
    int32        isSet      /* INPUT */
)
```

Parameters

cellId	Specifies the cell that owns the cell instance.
ciId	Specifies the cell instance to be set or reset as traceable.
isSet	Specifies if the cell instance is to be set or reset as traceable. Valid values: 1 means to set, 0 to reset.

Return

MWXSuccess	Occurs if the cell instance is successfully set or reset as traceable.
MWXFail	Occurs if the cell instance is not found.

Example

```
rc = MWXDb_Set_Net_Traceable(cellId, netId, 1);
switch (rc) {
    case MWXSuccess:
        return (MWX_TRUE_SCM);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to set net traceable\n");
        return (MWX_FALSE_SCM);
    default:
        MWX_Printf("ERROR -- unknown return code from MWXDb_Set_Net_Traceable\n");
        return (MWX_FALSE_SCM);
}
```

Scheme Equivalent

```
dbSetCellInstanceTraceById
```

MWXDb_Set_CellMustJoin2Pin

Establishes external connection specifications for two pins of a port.

Syntax

```
int MWXDb_Set_CellMustJoin2Pin (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t portId, /* INPUT */
    char *pinName1, /* INPUT */
    char *pinName2 /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell that owns the port and the pins.
portId	Specifies the port name.
pinName1	Specifies the name of the first pin.
pinName2	Specifies the name of the second pin.

Return

MWXNoLicense	Occurs if no software license is available.
MWXSuccess	Occurs if establishing must join specification is successful.
MWXFail	Occurs if establishing must join specification is not successful.

Example

```
rc = int MWXDb_Set_CellMustJoin2Pin( cellId, portId, pinName1, pinName2 )
switch (rc) {
    case MWXSuccess:
        return (MWX_TRUE_SCM);
    case MWXFail:
        return (MWX_FALSE_SCM);
}
```

Scheme Equivalent

dbSetCellMustJoin2Pin (see the *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Set_CellMustJoinPinPattern

Establishes external connection specification for all pins for a port uniquely identified by given portNamePattern.

Syntax

```
int MWXDb_Set_CellMustJoinPinPattern (
    MWXCellId_t cellId, /* INPUT */
    char *portNamePattern /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell.
portNamePattern	Specifies the pattern that uniquely identifies a port in the cell.

Return

MWXNoLicense	Occurs if no software license is available.
MWXSuccess	Occurs if the join patterns are set successfully or the pattern does not identify a unique port.
MWXFail	Occurs if the function fails to establish MustJoin for an identified port with multiple pins.

Example

```
rc = MWXDb_Set_CellMustJoinPinPattern( cellId,
    portNamePattern );
switch (rc) {
    case MWXSuccess:
return (MWX_TRUE_SCM);
    case MWXFail:
return (MWX_FALSE_SCM);
}
```

Scheme Equivalent

dbSetCellMustJoinPinPattern (see the *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Set_HorizontalWire_routeType

Sets the route type of a horizontal wire to a value in the 0–255 range.

Syntax

```
int MWXDb_Set_HorizontalWire_routeType(
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t  objId, /* INPUT */
    int32      value, /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell containing the wire.
objId	Specifies the database identifier of the wire for which the route type will be set.
value	Specifies the integer that represents a route type. Valid route types are in the range of 0–255.

Return

MWXSuccess	Occurs if the route type is successfully set.
MWXObjTyprMismatch	Occurs if the object ID does not represent a wire.
MWXInvalidArgumentValue	Occurs if the route type is out of valid range.
MWXInvalidCellIdOrObjId	Occurs if the cell ID or object ID is not valid.

Example

```
rc = MWXDb_Set_HorizontalWire_routeType (cellId, objId, value);
switch (rc) {
    case MWXSuccess:
        break;
    default:
        MWX_Printf("ERROR -- Failed to set routeType\n");
        break;
}
```

MWXDb_Set_Net_Traceable

Sets a net to be traceable or not traceable.

Syntax

```
int MWXDb_Set_Net_Traceable (
    MWXCellId_t cellId,    /* INPUT */
    MWXObjId_t  netId,     /* INPUT */
    int32       isSet      /* INPUT */
)
```

Parameters

cellId	Specifies the cell that owns the net.
netId	Specifies the net to be set or reset as traceable.
isSet	Specifies if the net is to be set or reset as traceable. Valid values: 1 means to set, 0 to reset.

Return

MWXSuccess	Occurs if the net is successfully set or reset as traceable.
MWXFail	Occurs if the net is not found.

Example

```
rc = MWXDb_Set_Net_Traceable(cellId, netId, 1);
switch (rc) {
    case MWXSuccess:
        return (MWX_TRUE_SCM);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to set net traceable\n");
        return (MWX_FALSE_SCM);
    default:
        MWX_Printf("ERROR -- unknown return code from MWXDb_Set_Net_Traceable\n");
        return (MWX_FALSE_SCM);
}
```

Scheme Equivalent

dbSetNetTraceById

MWXDb_Set_Netload

Sets the net load of the netPower to the database.

Syntax

```
int MWXDb_Set_NetLoad (
    MWXCellId_t cellId, /*INPUT*/
    MWXObjId_t netPowerId, /*INPUT*/
    float32 load /*INPUT*/
)
```


Parameters

cellId	Specifies the database identifier of the cell that owns the netPower object.
netPower	Specifies the database identifier of the netPower where the net load is going to be set.
load	Net load of the netPower object.

Return

MWXFail	Occurs if the net load is successfully set.
MWXSuccess	Occurs if the net load is not successfully set.

Example

```
rc = MWXDb_Set_NetLoad(cellId, netPowerId, &load);
    if (rc == MWXSuccess) {
        MWX_Printf("load : %lf\n", load);
    }
    else {
        MWX_Printf("MWXDb_Set_NetLoad fail, code = %d\n", rc);
        return (MWX_FALSE_SCM);
    }
    return (MWX_TRUE_SCM);
```

MWXDb_Set_NetSwitchActivity

Sets the switch activity of the netPower object in the database.

Syntax

```
int MWXDb_Set_NetSwitchActivity (
    MWXCellId_t    cellId,          /* INPUT */
    MWXObjId_t     netPowerId,      /* INPUT */
    int32          switchActivity   /* INPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell that owns the netPower object.
<code>netPowerId</code>	Specifies the database identifier of the netPower object where the net switch activity is going to be set.
<code>switchActivity</code>	Specifies the switch activity of the netPower object.

Return

<code>MWXFail</code>	Occurs if the net switch activity is successfully set.
<code>MWXSuccess</code>	Occurs if the net switch activity is not successfully set.

Example

```
rc = MWXDb_Set_NetSwitchActivity(cellId, netPowerId, &switchActivity);
    if (rc == MWXSuccess) {
        MWX_Printf("SwitchActivity : %d\n", switchActivity);
    }
    else {
        MWX_Printf("MWXDb_Set_NetSwitchActivity fail, code = %d\n", rc);
        return (MWX_FALSE_SCM);
    }
    return (MWX_TRUE_SCM);
```

MWXDb_Set_ScanChain_DefaultPriority

Sets the default priority for the scan chain.

Syntax

```
int MWXDb_Set_ScanChain_DefaultPriority (
    MWXCellId_t cellId, /* INPUT */
    char *scanChainName, /* INPUT */
    int32 priority /* INPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell that owns the scan chain.
<code>scanChainName</code>	Specifies the name of the scan chain for which the default priority is to be set.
<code>priority</code>	Default priority for the cell instances that are placed in an edge constraint but have not been assigned to a scan chain group. Valid values: an integer from 0 to 127 (0 is the lowest and 127 is the highest priority).

Return

<code>MWXNoLicense</code>	Occurs if no software license is available.
<code>MWXSuccess</code>	Occurs if the priority is successfully set.
<code>MWXFail</code>	Occurs if the priority cannot be successfully set.

Example

```
rc = MWXDb_Set_ScanChain_DefaultPriority( cellId,
                                         scanChainName
                                         priority );
switch (rc) {
    case MWXSuccess:
        return (MWX_TRUE_SCM);
    case MWXFail:
        return (MWX_FALSE_SCM);}
```

Scheme Equivalent

`dbSetScanChainDefaultPriority` (see the *Milkyway Environment Extension Language Reference Manual*)

MWXDb_Set_TopHierCellInst

Sets the topmost hierarchical cell instance.

Syntax

```
int MWXDb_Set_TopHierCellInst (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t hierCellInstId /* INPUT */
)
```

Parameters

<code>cellId</code>	Specifies the database identifier of the cell that is opened.
<code>hierCellInstId</code>	Specifies the database identifier of the topmost hierarchical cell instance that is retrieved.

Return

<code>MWXSuccess</code>	Occurs if the command successfully sets the topmost hierarchical cell instance.
<code>MWXFail</code>	Occurs if the command does not set the topmost hierarchical cell instance.

Example

```
int setTopHierCellInst (
    MWXCellId_t cellId,
    MWXObjId_t hierCellInstId
)
{
    int32 rc;
    rc = MWXDb_Set_TopHierCellInst(cellId, hierCellInstId);
    if (rc != MWXSuccess) {
        MWX_Printf((char*)"Fail to set topHierCellInst\n");
        return 0;
    }
    return 1;
}
```

See Also

[MWXDb_Get_TopHierCellInst](#)

MWXDb_Set_VerticalWire_routeType

Sets the route type of a vertical wire to a value in the 0–255 range.

Syntax

```
int MWXDb_Set_VerticalWire_routeType(
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t  objId, /* INPUT */
    int32       value, /* INPUT */
)
```

Parameters

cellId	Specifies the database identifier of the cell containing the wire.
objId	Specifies the database identifier of the wire for which the route type will be set.
value	Specifies the integer that represents a route type. Valid route types are in the 0–255 range.

Return

MWXSucccess	Occurs if the route type is successfully set.
MWXObjTypeMismatch	Occurs if the object ID does not represent a wire.
MWXInvalidArgumentValue	Occurs if the route type is out of valid range.
MWXInvalidCellIdOrObjId	Occurs if the cell ID or object ID is not valid.

Example

```
rc = MWXDb_Set_VerticalWire_routeType (cell_Id, obj_Id, value);
switch (rc) {
    case MWXSucccess:
        break;
    default:
        MWX_Printf("ERROR -- Failed to set routeType\n");
        break;
}
```

MWXDb_Translate_MilkywayName_ToVerilogName

Translates the Milkyway name in the input string, `inStr`, of the `libName` to the Verilog name stored in the output string, `outStr`.

Syntax

```
int MWXDb_Translate_MilkywayName_ToVerilogName(
    char *libName,      /* INPUT */
    char *inStr,        /* INPUT */
    char **outStr,      /* OUTPUT */
)
```

Parameters

<code>libName</code>	Specifies the database name of the library that contains the Milkyway name string to translate.
<code>inStr</code>	Specifies the Milkyway name string to translate.
<code>outStr</code>	Specifies the resulting Verilog name string.

Return

<code>MWXSuccess</code>	Occurs when the Milkyway name is successfully translated to the Verilog name.
<code>MWXFail</code>	Occurs when the Milkyway name fails to successfully translate to the Verilog name.

Example

```
rc = MWXDb_Translate_MilkywayName_ToVerilogName("temp", "a/c\\a_<5>" ,
&outStr);
switch (rc) {
    case MWXSuccess:
        MWX_Printf("Translated string is: %s \n", outStr);
        MWX_Printf("Translate Input String Successfully \n");
        return (MWX_TRUE_SCM);
    case MWXFail:
        MWX_Printf("ERROR -- Failed to translate\n");
        return (MWX_FALSE_SCM);
    default:
        MWX_Printf("ERROR -- unknown return code from
MWXDb_Translate_MilkywayName_ToVerilogName\n");
        return (MWX_FALSE_SCM);
}
```

Scheme Equivalent

`dbMilkywayTranslateString` (see the *IC Layout Command Reference Manual*)

MWXDb_Translate_VerilogName_ToMilkywayName

Translates the Verilog name in the input string, `inStr`, of the `libName` to the Milkyway name stored in the output string, `outStr`.

Syntax

```
int MWXDb_Translate_VerilogName_ToMilkywayName(  
    char *libName,      /* INPUT */  
    char *inStr,        /* INPUT */  
    char **outStr,      /* OUTPUT */  
)
```

Parameters

libName	Specifies the database name of the library that contains the Verilog name string to translate.
inStr	Specifies the Verilog name string to translate.
outStr	Specifies the resulting Milkyway name string.

Return

MWXSuccess	Occurs when the Verilog name is successfully translated to the Milkyway name.
MWXFail	Occurs when the Verilog name fails to successfully translate to the Milkyway name.

Example

```
rc = MWXDb_Translate_VerilogName_ToMilkywayName("temp", "\\a/b/e /c/d\\f/g /j/  
k" , &outStr);  
switch (rc) {  
    case MWXSuccess:  
        MWX_Printf("Translated string is: %s \n", outStr);  
        MWX_Printf("Translate Input String Successfully \n");  
        return (MWX_TRUE_SCM);  
    case MWXFail:  
        MWX_Printf("ERROR -- Failed to translate\n");  
        return (MWX_FALSE_SCM);  
    default:  
        MWX_Printf("ERROR -- unknown return code from  
        MWXDb_Translate_VerilogName_ToMilkywayName\n");  
        return (MWX_FALSE_SCM);  
}
```

Scheme Equivalent

dbVerilogTranslateString (see *IC Layout Command Reference Manual*)

Beginning with the V-2004.06 release, the following variable-width routing rule functions are available to the CAPI. You can use these functions to define rules, assign rule sets to nets, delete rules, and query or dump rules to a file without having to rely on the Scheme environment dbVarRouteRule functions.

MWXdB_Undelete_Object

Undeletes the object identified by `objectId` for the given cell.

Syntax

```
int MWXdB_Iint MWXdB_Undelete_Object (
    MWXCellId_t cellId, /* INPUT */
    MWXObjId_t objectId /* INPUT */
)
```

Parameters

`cellId` Specifies the database identifier of the cell from which you want to undelete the object.

`objectId` Specifies the database identifier of the object to undelete.

Return

`MWXSucces` Occurs if the object was successfully undeleted.

`MWXFail` Occurs if the object was not undeleted.

Dynamic Example

```
#include<stdio.h>
#include <stdlib.h>
#include "mwX.h"
#include "mwX_proto.h"

#ifdef __cplusplus
extern "C" {
    MWXSCH_t undeleteObject();
    void reg_undeleteObject();
}
#endif

MWXSCH_t undeleteObject (MWXSCH_t list)
{
    char progDescrString[50] = "MW-CAPI Example undeleteObject
-";
    static int caseSensitive = 0;
    int32 layer;
    MWXCellId_t cellId;
    int rc;
    int i,j;
    MWXPoint_t pointArr[64];
    MWXObjId_t rectId;
```



```

/*-----
HELLO
-----*/
MWX_Printf((char*)" \n%s ... \n", progDescrString);
if (MWXSc_Scanf (list, "o", &cellId) != 1) {
    MWX_Printf((char*)"Error.Synopsis : undeleteObject-api
cellId \n");
    return (MWX_FALSE_SCM);
}

MWX_Printf("cellId is %d\n", cellId);

/*-----
 * creating a rectangle
 *-----*/

pointArr[0].x = 500; pointArr[0].y = 500;
pointArr[1].x = 600; pointArr[1].y = 600;

rc = MWXDb_Create_Rectangle(cellId, "MlR", pointArr,
&rectId);
if (rc == MWXSuccess ) {
    MWX_Printf("Succeeded to create rectangle(%d)\n",
rectId);
}
else {
    MWX_Printf("Failed to create rectangle(%d)\n", rectId);
    return (MWX_FALSE_SCM);
}

/*-----
 * deleting a rectangle
 *-----*/

rc = MWXDb_Delete_Object(cellId,rectId);
if (rc == MWXSuccess) {
    MWX_Printf("Succeeded to delete
rectangle(%d)\n",rectId);
}
else {
    MWX_Printf("Failed to delete rectangle(%d)\n",rectId);
    return (MWX_FALSE_SCM);
}

/*-----
 * checking the status
 *-----*/
rc = MWXDb_Get_Rectangle_layer (cellId, rectId, &layer);
if (rc == MWXSuccess) {
    MWX_Printf("delete function is not working. layer is
%d\n",layer);
    return (MWX_FALSE_SCM);
}

```

```

    else {
        MWX_Printf("Successfull delete rectangle(%d). Unable
to get layer\n",rectId);
    }

    /*-----
    * undeleting a rectangle
    *-----*/

    rc = MWXDb_Undelete_Object (cellId, rectId);
    if (rc == MWXSuccess) {
        MWX_Printf("Succeeded to undelete
rectangle(%d)\n",rectId);
    }
    else {
        MWX_Printf("Failed to undelete rectangle(%d)\n",rectId);
        return (MWX_FALSE_SCM);
    }

    /*-----
    * checking the status
    *-----*/
    rc = MWXDb_Get_Rectangle_layer (cellId, rectId, &layer);
    if (rc == MWXSuccess) {
        MWX_Printf("layer is %d\n",layer);
    }
    else {
        MWX_Printf("Failed to undelete rectangle(%d). Unable
to get layer\n",rectId);
        return (MWX_FALSE_SCM);
    }

    /*-----
    DONE
    -----*/

    MWX_Printf((char*) "%s GOODBYE! \n\n",progDescrString);
    return (MWX_TRUE_SCM);
}

void reg_undeleteObject(){
    MWXSc_Register_Function((char*) "undeleteObject-
api",undeleteObject);
}

```

See Also

[MWXDb_Delete_Object](#)

9

PARA View Commands

This chapter describes Milkyway parasitic data layer commands, which provide efficient access to Milkyway PARA view data. The PARA commands access the PARA view objects and field data. For more information about objects and fields, see [Chapter 3, “Database Objects, Object Fields, and Field Accessor Functions”](#) and Appendix A, “Milkyway C-API List of Functions.”

These PARA view commands use a runtime parasitic data layer application to encapsulate the internal representation of parasitic data. All parasitic information is loaded into this runtime data layer, which you access through the set of C-API commands as described in this chapter.

See [“Dynamic PARA View Pvx* Example” on page 9-35](#) for an example that uses the PARA view commands and shows the initialization of a design framework and the creation of a parasitic runtime data structure.

Dynamic PARA View Pvx* Examples

The following examples demonstrate the use of the PARA View Pvx* commands.

- [Example With PARA Commands](#)
- [Example With Pvx* Commands](#)

Example With PARA Commands

[Example 9-1](#) uses the PARA view commands described in [Chapter 3, “Database Objects, Object Fields, and Field Accessor Functions.”](#) These functions work at the segment level and do not use a runtime parasitic data layer application to encapsulate the internal representation of parasitic data. The example also demonstrates the use of iteration functions.

Example 9-1 Para Commands

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include "mwX.h"
#include "mwX_proto.h"
#include "omwX_proto.h"
void free_ObjId_memory (MWXObjId_t *ptr) {
    if (ptr) {
        free(ptr);
    }
}

/* Function to Retrieve Data for RLCNode's attributes */
void Get_RLCNode_Data(MWXCellId_t cellId, MWXObjId_t objId, FILE *outFile) {
    uint32 nodeId;
    float nomCap;
    int rc; /* returned Code */
    rc = MWXDb_Get_RLCNode_nodeId(cellId,objId,&nodeId);

    if (rc != MWXSuccess) {
        MWX_Printf("Fail to get nodeId of RLCNode object\n");
        exit(MWXFail);
    }

    rc = MWXDb_Get_RLCNode_nomCap(cellId,objId,&nomCap);

    if (rc != MWXSuccess) {
        MWX_Printf("Fail to get nomCap in RLCNode object\n");
        exit(MWXFail);
    }

    fprintf(outFile,"%-10d %g\n",nodeId,nomCap);
}

/* Function to Retrieve Data for REdge's attributes */
```

```

void Get_RLCEdge_Data (MWXCellId_t cellId, MWXObjId_t objId, FILE *outFile){
    uint32 nodeId1, nodeId2;
    float nomRes;
    int rc; /* returned Code */
    rc = MWXDb_Get_REdge_nodeId1(cellId,objId,&nodeId1);

    if (rc != MWXSuccess) {
        MWX_Printf("Fail to get nodeId1 of REdge object\n");
        exit(MWXFail);
    }

    rc = MWXDb_Get_REdge_nodeId2(cellId,objId,&nodeId2);

    if (rc != MWXSuccess) {
        MWX_Printf("Fail to get nodeId2 of REdge object\n");
        exit(MWXFail);
    }

    rc = MWXDb_Get_REdge_nomRes(cellId,objId,&nomRes);

    if (rc != MWXSuccess) {
        MWX_Printf("Fail to get nomRes in REdge object\n");
        exit(MWXFail);
    }

    fprintf(outFile,"%-10d %-10d %g \n",nodeId1,nodeId2,nomRes);
}

/* Function to Retrieve Data for CEdge's attributes */
void Get_CEdge_Data (MWXCellId_t cellId,MWXObjId_t objId, FILE *outFile){
    uint32 nodeId1, nodeId2; float nomCap;
    int rc; /* returned Code */
    rc = MWXDb_Get_CEdge_nodeId1(cellId,objId,&nodeId1);

    if (rc != MWXSuccess) {
        MWX_Printf("Fail to get nodeId1 of CEdge object\n");
        exit(MWXFail);
    }

    rc = MWXDb_Get_CEdge_nodeId2(cellId,objId,&nodeId2);

    if (rc != MWXSuccess) {
        MWX_Printf("Fail to get nodeId2 of CEdge object\n");
        exit(MWXFail);
    }

    rc = MWXDb_Get_CEdge_nomCap(cellId,objId,&nomCap);

    if (rc != MWXSuccess) {
        MWX_Printf("Fail to get nomCap in CEdge object\n");
        exit(MWXFail);
    }

    fprintf(outFile,"%-10d %-10d %g \n",nodeId1,nodeId2,nomCap);
}

/* Function to Retrieve Data for LEdge's attributes */
void Get_LEdge_Data (MWXCellId_t cellId,MWXObjId_t objId, FILE *outFile){

```

```

uint32 nodeId1, nodeId2;
float nomInd;
int rc; /* returned Code */

rc = MWXDb_Get_LEdge_nodeId1(cellId,objId,&nodeId1);

if (rc != MWXSuccess) {
    MWX_Printf("Fail to get nodeId1 of LEdge object\n");
    exit(MWXFail);
}

rc = MWXDb_Get_LEdge_nodeId2(cellId,objId,&nodeId2);

if (rc != MWXSuccess) {
    MWX_Printf("Fail to get nodeId2 of LEdge object\n");
    exit(MWXFail);
}

rc = MWXDb_Get_LEdge_nomInd(cellId,objId,&nomInd);

if (rc != MWXSuccess) {
    MWX_Printf("Fail to get nomRes in LEdge object\n");
    exit(MWXFail);
}

fprintf(outFile,"%-10d %-10d %g \n",nodeId1,nodeId2,nomInd);
}

/* Function to Retrieve Data for RLCPort's attributes */
void Get_RLCPort_Data ( MWXCellId_t cellId, MWXObjId_t objId, FILE *outFile,
                        uint32 *portId, MWXObjId_t *cellInstId){
    MWXObjId_t instId, nodeObjId;
    uint32 nodeId;
    uint32 portNameOrObjId;
    int rc; /* returned Code */

    rc = MWXDb_Get_RLCPort_instId(cellId,objId,&instId);

    if (rc != MWXSuccess) {
        MWX_Printf("Fail to get instId of RLCPort object\n");
        exit(MWXFail);
    }

    rc = MWXDb_Get_RLCPort_portNameOrObjId(cellId,objId,&portNameOrObjId);

    if (rc != MWXSuccess) {
        MWX_Printf("Fail to get portNameOrObjId in RLCPort object\n");
        exit(MWXFail);
    }

    *portId = portNameOrObjId;
    *cellInstId = instId;
}

/*----- Main function -----*/

int main( int argc,char **argv) {
    char *progDescrString = "PARA View Verification Test -";

```

```

static int caseSensitive = 0;
char cellName[1024],paraCellName[1024],netName[1024];
MWXLibId_t libId;
MWXCellId_t cellId, cellId2;
MWXObjId_t *RLCNodeId, *REdgeId, *CEdgeId, *LEdgeId, *RLCPortId,
    *RLCNetId, *RLCCoupleNetId;
int32 pCount,objCount;
FILE *outFile,*outFile2,*outFile3,*outFile4;
MWXObjId_t netId,instId,objId,nodeObjId,cEdgeId;
uint32 portNameOrObjId;
MWXObjId_t *portInstId;
MWXObjId_t cellInstId,pIMId;
MWXObjTypeId_t objType;
MWXIteratorId_t portIterId,nodeIterId, rEdgeIterId, coupleNetIterId,
    cEdgeIterId;
char *instName,*termName;
int rc; /*returned Code */
int i,j;
int match = 0;
uint32 nodeId;
float nomCap;

/*----- HELLO -----*/

    MWX_Printf("\n%s ... \n", progDescrString);

/*----- Initialize Open Milkyway -----*/

    MWX_Printf("%s Calling MWXDb_omwxInit to INITIALIZE MILKYWAY ... \n",
        progDescrString);
    rc = MWXDb_omwxInit(argc, argv, caseSensitive); /* argc argv used in lmCO */

    switch(rc) {
        case MWXSuccess:
            MWX_Printf("%s SUCCESSFUL : INITIALIZATION ... \n", progDescrString);
            break;
        default:
            MWX_Printf("%s *** ERROR - BAD INITIALIZATION.\n", progDescrString);
            exit(MWXFail);
    }

    if (argc != 5) {
        MWX_Printf("Usage: PARAvue libName fullCelViewCellName
            fullParaViewCellName netName\n");
        exit (MWXFail);
    }

    sprintf(cellName,"%s",argv[2]);
    sprintf(paraCellName,"%s",argv[3]);
    sprintf(netName,"%s",argv[4]);

/*----- OPEN LIBRARY -----*/

    MWX_Printf("%s Opening LIBRARY %s USING MWXDb_Open_Li ... \n",
        progDescrString, argv[1]);
    rc = MWXDb_Open_Lib(argv[1],&libId);

    switch(rc) {

```

```

        case MWXSuccess:
            MWX_Printf("%s SUCCESSFUL : LIBRARY OPEN.\n", progDescrString);
            break;
        case MWXFail:
            MWX_Printf("%s *** ERROR - Library Open Failure\n",progDescrString);
            exit(MWXFail);
        case MWXNoLicense:
            MWX_Printf("%s *** ERROR - NO LICENSE\n", progDescrString);
            exit(MWXFail);
        default: MWX_Printf("%s *** UNKNOWN ERROR %d from MWXDb_Open_Lib\n",
            progDescrString, rc);
            exit(MWXFail);
    }

    MWX_Printf("%s --- OPENED LIBRARY ID is %d\n",progDescrString,libId);

/*----- Open cell with CEL view and PARA view -----*/

    MWX_Printf("%s Trying to get open cell %s ...\n",
        progDescrString, cellName);
    rc = MWXDb_Open_Cell(cellName,'r",&cellId);

    if (rc!=MWXSuccess) {
        MWX_Printf("Unable to open this cell %s\n",cellName);
        exit(MWXFail);
    }

    MWX_Printf("%s Trying to get open cell %s ...\n",
        progDescrString,paraCellName);
    rc = MWXDb_Open_Cell(paraCellName,'r",&cellId2);

    if (rc!=MWXSuccess) {
        MWX_Printf("Unable to open this cell
%s\n",paraCellName);
        exit(MWXFail);
    }

    outFile = fopen ("portOutFile", "w");

    if (outFile == NULL) {
        fprintf (stderr, "Fail to open outFile\n");
        exit(1);
    }

    outFile2 = fopen ("nodeOutFile", "w");

    if (outFile2 == NULL) {
        fprintf (stderr, "Fail to open outFile2\n");
        exit(1);
    }

    outFile3 = fopen ("rEdgeOutFile", "w");

    if (outFile3 == NULL) {
        fprintf (stderr, "Fail to open outFile3\n");
        exit(1);
    }
}

```



```

outFile4 = fopen ("cEdgeOutFile", "w");

if (outFile4 == NULL) {
    fprintf (stderr, "Fail to open outFile4\n");
    exit(1);
}

fprintf(outFile,"NodeId      Instance Name      TermName\n");
fprintf(outFile2,"NodeId      nomCap\n");
fprintf(outFile3,"NodeId1      NodeId2      nomRes\n");
fprintf(outFile4,"NodeId1      NodeId2      nomCap\n");

rc = MWXDb_Get_Objects_ByType(cellId2, MWX_RLCNET, &objCount, &RLCNetId);

if (rc != MWXSuccess) {
    MWX_Printf("Fail to get RLCNet object Ids in cell %s\n",cellName);
    exit(MWXFail);
}

if (objCount == 0) {
    MWX_Printf("There is none RLCNet object in this cell\n");
}

else {
    for (i = 0; i< objCount; i++) {
        rc = MWXDb_Get_Net_ByName(cellId,netName,&netId);

        if(rc!=MWXSuccess) {
            MWX_Printf("Fail to get net by name\n");
            exit(MWXFail);
        }

        rc = MWXDb_Get_NetPortInst(cellId,netId,&pCount,&portInstId);

        if(rc!=MWXSuccess) {
            MWX_Printf("Fail to get portInst from net\n");
            exit(MWXFail);
        }

        if (pCount == 0) {
            MWX_Printf("There is no portInst belongs to this net\n");
        }

        else {
            for(j=0;j<pCount;j++){
/*----- TESTING ACCESS DATA of RLCNet object -----*/

                rc = MWXDb_Get_PortInst_cellInstId(cellId,portInstId[j],
                    &cellInstId);

                if (rc != MWXSuccess) {
                    MWX_Printf("Fail to get cellInstId from portInst\n");
                    exit(MWXFail);
                }

                if (MWXDb_Get_CellInst_name(cellId,cellInstId,&instName) !=
                    MWXSuccess){

```

```

        MWX_Printf("Fail to get cellInst name\n");
        exit(MWXFail);
    }

    if (MWXDb_Get_PortInst_portInstMasterId(cellId,
        portInstId[j], &pIMId) != MWXSuccess) {
        MWX_Printf("Fail to get portInstMasterId from portInst\n");
        exit(MWXFail);
    }

    if (MWXDb_Get_PortInstMaster_name(cellId,pIMId,&termName) !=
        MWXSuccess){
        MWX_Printf("Fail to get portInstMaster name\n");
        exit(MWXFail);
    }

/*----- TESTING GET RLCPORT information from netName -----*/
    rc = MWXDb_Iter_RLCNetToPort(cellId2,*(RLCNetId+i),
        &portIterId);

    if (rc != MWXSuccess) {
        MWX_Printf("Fail to Iter RLCNet To Port\n");
        exit(MWXFail);
    }

    while ((rc = MWXDb_Iter_Next(portIterId,&objId)) !=
        MWXIterDone) {

        if (rc == MWXFail) {
            MWX_Printf("Fail to do RLCPort iteration\n");
            continue;
        }

        Get_RLCPort_Data(cellId2,objId,outFile,&portNameOrObjId,
            &instId);

        if((instId == cellInstId) && (portNameOrObjId ==
            portInstId[j])) {
            match = 1;
            rc = MWXDb_Get_RLCPort_nodeObjId(cellId2,objId,
                &nodeObjId);
            if (rc != MWXSuccess) {
                MWX_Printf("Fail to get RLCPort nodeObjId\n");
                continue;
            }

            MWX_Printf("nodeObjId is %d\n",nodeObjId);

            if (MWXDb_Get_RLCNode_nodeId(cellId2,nodeObjId,&nodeId)
                != MWXSuccess){
                MWX_Printf("Fail to get RLCNode nodeId\n");
                continue;
            }
            printf(outFile,"%-12d %-50s %s\n", nodeId, instName,
                termName);
        }
    }
    MWXDb_Iter_Quit(portIterId);

```

```

    }

    if (pCount) free_ObjId_memory(portInstId);
}
/*----- TESTING GET RLCNODE information from netName -----*/

if (match) {
    rc = MWXDb_Iter_RLCNetToNode(cellId2, *(RLCNetId+i),&nodeIterId);

    if (rc != MWXSuccess) {
        MWX_Printf("Fail to Iter RLCNet To Node\n");
        exit(MWXFail);
    }

    while ((rc = MWXDb_Iter_Next(nodeIterId,&objId)) != MWXIterDone) {

        if (rc == MWXFail) {
            MWX_Printf("Fail to do RLCNode iteration\n");
            continue;
        }

        Get_RLCNode_Data(cellId2,objId,outFile2);
    }

    MWX_Printf("Done with the node iteration\n");
MWXDb_Iter_Quit(nodeIterId);

/*----- TESTING GET RLCEdge information from netName -----*/

    rc = MWXDb_Iter_RLCNetToREdge(cellId2,*(RLCNetId+i),&rEdgeIterId);

    if (rc != MWXSuccess) {
        MWX_Printf("Fail to Iter RLCNet To REdge\n");
        exit(MWXFail);
    }

    while ((rc = MWXDb_Iter_Next(rEdgeIterId,&objId)) != MWXIterDone) {

        if (rc == MWXFail) {
            MWX_Printf("Fail to do RLCEdge iteration\n");
            continue;
        }

        Get_RLCEdge_Data(cellId2,objId,outFile3);
    }
    MWXDb_Iter_Quit(rEdgeIterId);

/*----- TESTING GET RLCCoupleNet information from netName -----*/

    rc = MWXDb_Iter_RLCNetToCoupleNet(cellId2,*(RLCNetId+i),
        &coupleNetIterId);

    if (rc != MWXSuccess) {
        MWX_Printf("Fail to Iter RLCNet To RLCCoupleNet\n");
        exit(MWXFail);
    }

    while ((rc = MWXDb_Iter_Next(coupleNetIterId,&objId)) !=

```

```

        MWXIterDone) {

    if (rc == MWXFail) {
        MWX_Printf("Fail to do RLCCoupleNet iteration\n");
        continue;
    }

    rc = MWXDb_Iter_RLCCoupleNetToCEdge(cellId2,objId,
        &cEdgeIterId);

    if (rc != MWXSuccess) {
        MWX_Printf("Fail to Iter RLCCoupleNet to RLCCEdge\n");
        continue;
    }

    while((rc = MWXDb_Iter_Next(cEdgeIterId,&cEdgeId)) !=
        MWXIterDone) {

        if (rc == MWXFail) {
            MWX_Printf("Fail to do RLCCEdge iteration\n");
            continue;
        }

        Get_CEEdge_Data(cellId2,cEdgeId,outFile4);
    }
    MWXDb_Iter_Quit(cEdgeIterId);
}
    MWXDb_Iter_Quit(coupleNetIterId);
}
    match = 0;
}

    if (objCount) free_ObjId_memory(RLCNetId);
}

rc = MWXDb_Close_Cell(cellId);

if (rc != MWXSuccess) {
    MWX_Printf("%s *** ERROR - Unable to close cell\n",progDescrString);
    exit(MWXFail);
}

rc = MWXDb_Close_Cell(cellId2);

if (rc != MWXSuccess) {
    MWX_Printf("%s *** ERROR - Unable to close cell\n",progDescrString);
    exit(MWXFail);
}

fclose(outFile);
fclose(outFile2);
fclose(outFile3);
fclose(outFile4);

rc = MWXDb_Close_Lib();

switch(rc) {
    case MWXSuccess:

```

```

        MWX_Printf("%s SUCCESSFUL : LIBRARY CLOSE.\n", progDescrString);
        break;
    case MWXNoLibraryOpened:
        MWX_Printf("%s *** ERROR - NO LIBRARY OPENED\n",progDescrString);
        exit(MWXFail);
    case MWXFail:
        MWX_Printf("%s *** ERROR - Library Close Failure\n", progDescrString);
        exit(MWXFail);
    default:
        MWX_Printf("%s *** UNKNOWN ERROR %d from MWXDb_Close_Lib\n",
            progDescrString,rc);
        exit(MWXFail);
}

/*----- DONE -----*/

MWX_Printf("%s GOODBYE! \n\n",progDescrString);
exit(MWXSuccess);
}

```

Example With Pvx* Commands

[Example 9-2](#) uses Pvx* commands and shows the initialization of a design framework and the creation of a parasitic runtime data structure. After constructing an RLC tree for the specified net, the corresponding data is obtained by traversing several object types. The example also shows the accessing of coupling mesh information. The design is then closed to free allocated memory.

Example 9-2 Pvx* Commands

```

#include <stdlib.h>
#include <malloc.h>
#include <assert.h>
#include "mwX.h"

#ifdef __cplusplus
#include "mwX_proto.h"
extern "C" {
    MWXSCM_t PvxDesignTest();
    void mwInitPvxDesignTest();
}
#endif

MWXSCM_t mwPvxDesignTest (MWXSCM_t list)
{
    void *pvxDesign = NULL;
    char libName[256], pvCellName[256], cvCellName[256];
    int rc;
    MWXPvxObjData_t rlcNetWrap;
    MWXPvxObjData_t rlcPortWrap, rlcNodeWrap, rlcEdgeWrap, rlcCoupleNetWrap;
    MWXPvxObjData_t aggNetWrap, cEdgeWrap, aggNodeWrap;
    MWXPvxRLCNetField_t dataType;
    MWXPvxRLCCoupleNetField_t cDataType;
    char strNetName[256];
    char *netName;
    int numOfNode, numOfEdge;
    int portNodeId;

```

```

char *instName, *termName;
int rlcType, nodeType, rlcNodeId;
float nomCap, minCap, maxCap;
int count;
int rlcNodeId1, rlcNodeId2;
float nomRes, minRes, maxRes;
int vicNodeId, aggNodeId;
float nomCouple, minCouple, maxCouple;
MWXPvxObjData_t net1Wrap, net2Wrap;

if (MWXSc_Scanf(list, (char*) "ssss", libName, pvCellName,
    cvCellName, strNetName) != 4) {
    MWX_Printf((char*) "ERROR -- mwPvxDesignTest requires 4 arguments\n");
    return (MWX_FALSE_SCM);
}

if (MWXDb_Init_PvxDesign( libName, pvCellName, cvCellName,
    &pvxDesign) != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to initialize the para design\n");
    return (MWX_FALSE_SCM);
}

if (MWXDb_Create_PvxRCNetworkForAllNets( pvxDesign) != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to create RC network\n");
    return (MWX_FALSE_SCM);
}

if (MWXDb_Get_PvxRLCNet_ByName( pvxDesign, strNetName, &rlcNetWrap) !=
    MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to get RLCNet handle for net %s\n",
        strNetName);
    return (MWX_FALSE_SCM);
}

if (MWXDb_Get_PvxRLCNetInfo( pvxDesign, &rlcNetWrap, &netName, &numOfNode,
    &numOfEdge) != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to get RLCNet information\n");
    return (MWX_FALSE_SCM);
}

MWX_Printf((char*) "\nNet %s    numOfNode = %d    numOfEdge = %d\n", netName,
    numOfNode, numOfEdge);

if (MWXDb_Init_PvxRLCNetIteration( pvxDesign, &rlcNetWrap) != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to initialize RLCNet iteration\n");
    return (MWX_FALSE_SCM);
}

/* Port */
dataType = MWXPvxRLCPortType;

if (MWXDb_Iter_PvxRLCNetMember_ByType( &rlcNetWrap, dataType) !=
    MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to start iteration for
        RLCPort
type\n");
    return (MWX_FALSE_SCM);
}

```

```

while ((rc = MWXDb_Iter_Next_PvxRLCNetMember_ByType( &rlcNetWrap, dataType,
&rlcPortWrap)) != MWXIterDone) {

    if (rc == MWXFail) {
        MWX_Printf((char*) "ERROR -- Failed to iterate next PvxRLCNetMember\n");
        continue;
    }

    if (MWXDb_Get_PvxRLCPortInfo( pvxDesign, &rlcPortWrap, &portNodeId,
        &instName, &termName) != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to get RLCPort data\n");
        continue;
    }

    MWX_Printf((char*) "*I %s/%s    nodeId: %d\n", instName, termName,
portNodeId);
}
/* End of Port */

/* Node */
dataType = MWXPvxRLCNodeType;

if (MWXDb_Iter_PvxRLCNetMember_ByType( &rlcNetWrap, dataType) !=
    MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to start iteration for
        RLCNode type\n");
    return (MWX_FALSE_SCM);
}

MWX_Printf((char*) "Cap section\n");
rlcType = MWXCpvxRLCTree;
count = 0;
while ((rc = MWXDb_Iter_Next_PvxRLCNetMember_ByType( &rlcNetWrap,
    dataType, &rlcNodeWrap)) != MWXIterDone) {

    if (rc == MWXFail) {
        MWX_Printf((char*) "ERROR -- Failed to iterate next PvxRLCNetMember\n");
        continue;
    }

    if (MWXDb_Get_PvxRLCNodeInfo( pvxDesign, &rlcNodeWrap, rlcType,
        &nodeType, &rlcNodeId, &nomCap, &minCap, &maxCap) != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to get RLCNode data\n");
        continue;
    }

    MWX_Printf((char*) "%d    id:%d (%g %g %g)\n", count++, rlcNodeId,
        nomCap, minCap, maxCap);
}
/* End of Node */

/* Edge */
dataType = MWXPvxRLCEdgeType;

if (MWXDb_Iter_PvxRLCNetMember_ByType( &rlcNetWrap, dataType) !=
    MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to start iteration for

```

```

        RLCEdge type\n");
    return (MWX_FALSE_SCM);
}

MWX_Printf((char*) "Res section\n");
count = 0;
while ((rc = MWXDb_Iter_Next_PvxRLCNetMember_ByType( &rlcNetWrap,
    dataType, &rlcEdgeWrap)) != MWXIterDone) {

    if (rc == MWXFail) {
        MWX_Printf((char*) "ERROR -- Failed to iterate next PvxRLCNetMember\n");
        continue;
    }

    if (MWXDb_Get_PvxRLCEdgeInfo( pvxDesign, &rlcEdgeWrap, &rlcNodeId1,
        &rlcNodeId2, &nomRes, &minRes, &maxRes) != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to get RLCEdge data\n");
        continue;
    }

    MWX_Printf((char*) "%d  %d ---> %d (%g %g %g)\n", count++,
        rlcNodeId1, rlcNodeId2, nomRes, minRes, maxRes);
}
/* End of Edge */

/* Couple Net */
dataType = MWXPvxRLCCoupleNetType;

if (MWXDb_Iter_PvxRLCNetMember_ByType( &rlcNetWrap, dataType) !=
    MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to start iteration for couple net
        type\n");
    return (MWX_FALSE_SCM);
}
MWX_Printf((char*) "Couple section\n");
count = 0;
while ((rc = MWXDb_Iter_Next_PvxRLCNetMember_ByType( &rlcNetWrap, dataType,
    &rlcCoupleNetWrap)) != MWXIterDone) {

    if (rc == MWXFail) {
        MWX_Printf((char*) "ERROR -- Failed to iterate next PvxRLCNetMember\n");
        continue;
    }

    if (MWXDb_Get_PvxRLCCoupleNetInfo( pvxDesign, &rlcCoupleNetWrap,
        &net1Wrap, &net2Wrap) != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to get net1/net2 data\n");
    }

    if (MWXDb_Get_PvxRLCNetInfo( pvxDesign, &net1Wrap, &netName,
        &numOfNode, &numOfEdge) != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to get net1 name\n");
        return (MWX_FALSE_SCM);
    }

    MWX_Printf((char*) "Net 1:  netName=%s  numOfNode=%d  numOfEdge=%d \n",
        netName, numOfNode, numOfEdge);
}

```



```

    if (MWXDb_Get_PvxRLCNetInfo( pvxDsgn, &net2Wrap, &netName,
        &numOfNode, &numOfEdge) != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to get net2 name\n");
        return (MWX_FALSE_SCM);
    }

    MWX_Printf((char*) "Net 2:  netName=%s    numOfNode=%d    numOfEdge=%d \n",
        netName, numOfNode, numOfEdge);

    if (MWXDb_Get_PvxRLCCoupleNetAggNetInfo( pvxDsgn, &rlcNetWrap,
        &rlcCoupleNetWrap, &aggNetWrap) != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to get aggressor net data\n");
        continue;
    }

    if (MWXDb_Get_PvxRLCNetInfo( pvxDsgn, &aggNetWrap, &netName,
        &numOfNode, &numOfEdge) != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to get aggressor net name\n");
        return (MWX_FALSE_SCM);
    }

    MWX_Printf((char*) "Agg netName : %s\n", netName);

    if (MWXDb_Init_PvxRLCCoupleNetIteration( pvxDsgn, &rlcCoupleNetWrap) !=
        MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to initialize RLCCoupleNet
            iteration\n");
        return (MWX_FALSE_SCM);
    }

    cDataType = MWXPvxCEdgeType;
    while ((rc = MWXDb_Iter_Next_PvxRLCCoupleNetMember_ByType(
        &rlcCoupleNetWrap, cDataType, &cEdgeWrap)) != MWXIterDone) {

        if (rc == MWXFail) {
            MWX_Printf((char*) "ERROR -- Failed to iterate next
                PvxRLCCoupleNetMember\n");
            continue;
        }

        if (MWXDb_Get_PvxCEdgeInfo( pvxDsgn, &cEdgeWrap, &rlcNetWrap,
            &aggNodeWrap, &vicNodeId, &aggNodeId, &nomCouple, &minCouple,
            &maxCouple) != MWXSuccess) {
            MWX_Printf((char*) "ERROR -- Failed to get CEdge data\n");
            continue;
        }

        MWX_Printf((char*) "%d  %d ---> %d (%g %g %g)\n", count++, vicNodeId,
            aggNodeId, nomCouple, minCouple, maxCouple);
    }

    if (MWXDb_Close_PvxRLCCoupleNetIteration( pvxDsgn, &rlcCoupleNetWrap)
        != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to terminate RLCCoupleNet
            iteration\n");
        return (MWX_FALSE_SCM);
    }
}

```

```

/* End of Couple Net */

if (MWXDb_Close_PvxRLCNetIteration( pvxDesign, &rlcNetWrap) != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to terminate RLCNet iteration\n");
    return (MWX_FALSE_SCM);
}

if (MWXDb_Close_PvxDesign( pvxDesign) != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to close the para design\n");
    return (MWX_FALSE_SCM);
}

return (MWX_TRUE_SCM);
}

/* Define a function to register the new command. */
void mwInitPvxDesignTest(){
    MWXSc_Register_Function ((char*) "mwPvxDesignTest", mwPvxDesignTest);
}

```

MWXDb_Close_PvxDesign

Closes the runtime parasitic data layer application and releases the memory. Note that you have to call this function at the same time as pvxDesign initialization.

Syntax

```

int MWXDb_Close_PvxDesign (
    void *pvxDesign
)

```

Parameters

pvxDesign	Specifies the handle of the parasitic design.

Return

MWXSuccess	Occurs if the PARA application successfully closes.
MWXFail	Occurs if the PARA application fails to close.

Example

```

if (MWXDb_Close_PvxDesign( pvxDesign) != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to close the para design\n");
    return (MWX_FALSE_SCM);
}

```

MWXdB_Close_PvxRLCCoupleNetIteration

Closes the RLCCoupleNet iteration and releases the memory.

Syntax

```
int MWXdB_Close_PvxRLCCoupleNetIteration (  
    void *pvxDesign, /* INPUT */  
    MWXPvxObjData_t *coupleNetWrap /* INPUT/OUTPUT */  
)
```

Parameters

pvxDesign	Specifies the handle of the PARA design.
coupleNetWrap	Specifies the record of the RLCCoupleNet iteration.

Return

MWXSuccess	Occurs if the RLCCoupleNet iteration successfully closes.
MWXFail	Occurs if the RLCCoupleNet iteration fails to close.

Example

```
    if (MWXdB_Close_PvxRLCCoupleNetIteration( pvxDesign,  
&rlcCoupleNetWrap)  
        != MWXSuccess) {  
        MWX_Printf((char*) "ERROR -- Failed to terminate RLCCoupleNet  
                           iteration\n");  
        return (MWX_FALSE_SCM);  
    }
```

MWXdB_Close_PvxRLCNetIteration

Closes the RLCNet iteration and releases the memory.

Syntax

```
int MWXdB_Close_PvxRLCNetIteration (  
    void *pvxDesign, /* INPUT */  
    MWXPvxObjData_t *rlcNetWrap /* INPUT/OUTPUT */  
)
```

Parameters

<code>pvxDesign</code>	Specifies the handle of the PARA design.
<code>rlcNetWrap</code>	Specifies the record of the RLCNet iteration.

Return

<code>MWXSuccess</code>	Occurs if the RLCNet iteration successfully closes.
<code>MWXFail</code>	Occurs if the RLCNet iteration fails to close.

Example

```
if (MWXDb_Init_PvxRLCNetIteration( pvxDesign, &rlcNetWrap) != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to initialize RLCNet iteration\n");
    return (MWX_FALSE_SCM);
}

{
    ...    /* data processing */
}

if (MWXDb_Close_PvxRLCNetIteration( pvxDesign, &rlcNetWrap) != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to terminate RLCNet iteration\n");
    return (MWX_FALSE_SCM);
}
```

MWXDb_Create_PvxRCNetworkForAllNets

Builds the parasitic data structure by reading the PARA view object RLCNet for all nets and adds the coupling mesh information between the nets by reading the RLCCoupleNet objects.

Syntax

```
int MWXDb_Create_PvxRCNetworkForAllNets (
    void *pvxDesign /* INPUT */
)
```

Parameters

`pvxDesign` Specifies the handle of the parasitic design.

Return

`MWXSucces` Occurs if the parasitic network is successfully created.

`MWXFail` Occurs if the parasitic network is not created.

Example

```
if (MWXDb_Init_PvxDesign( libName, pvCellName, cvCellName, &pvxDesign)
    != MWXSucces) {
    MWX_Printf((char*) "ERROR -- Failed to initialize the para
design\n");
    return (MWX_FALSE_SCM);
}

if (MWXDb_Create_PvxRCNetworkForAllNets( pvxDesign) != MWXSucces) {
    MWX_Printf((char*) "ERROR -- Failed to create RC network\n");
    return (MWX_FALSE_SCM);
}
```

MWXDb_Get_PvxCEdgeInfo

Gets the CEdge information.

Syntax

```
int MWXDb_Get_PvxCEdgeInfo (
    void *pvxDesign, /* INPUT */
    MWXPvxObjData_t *cEdgeWrap, /* INPUT */
    MWXPvxObjData_t *vicNetWrap, /* INPUT */
    MWXPvxObjData_t *aggNodeWrap, /* OUTPUT */
    int32 *vicNodeId, /* OUTPUT */
    int32 *aggNodeId, /* OUTPUT */
    float *nomCouple, /* OUTPUT */
    float *minCouple, /* OUTPUT */
    float *maxCouple /* OUTPUT */
)
```

Parameters

<code>pvxDesign</code>	Specifies the handle of the PARA design.
<code>cEdgeWrap</code>	Specifies the CEdge record.
<code>vicNetWrap</code>	Specifies the record of the victim net.
<code>vicNodeId</code>	Specifies the node ID of the victim.
<code>aggNodeId</code>	Specifies the node ID of the aggressor.
<code>nomCouple</code>	Specifies the nominal coupling.
<code>minCouple</code>	Specifies the minimum coupling.
<code>maxCouple</code>	Specifies the maximum coupling.

Return

<code>MWXSucces</code>	Occurs if the CEdge data is successfully returned.
<code>MWXFail</code>	Occurs if the CEdge data is not returned.

Example

```
MWXPvxObjData_t rlcCoupleNetWrap;
MWXPvxObjData_t cEdgeWrap, aggNodeWrap;

if (MWXDb_Init_PvxRLCCoupleNetIteration( pvxDesign, &rlcCoupleNetWrap)
    != MWXSucces) {
    MWX_Printf((char*) "ERROR -- Failed to initialize RLCCoupleNet
iteration\n");
    return (MWX_FALSE_SCM);
}

cDataType = MWXPvxCEdgeType;
while ((rc = MWXDb_Iter_Next_PvxRLCCoupleNetMember_ByType(
    &rlcCoupleNetWrap, cDataType, &cEdgeWrap)) != MWXIterDone){
    if(rc == MWXFail){
        MWX_Printf((char*)"ERROR -- Failed to iterate next PvxRLCCoupleNet\n");
        continue;
    }
    if (MWXDb_Get_PvxCEdgeInfo( pvxDesign, &cEdgeWrap, &rlcNetWrap,
        &aggNodeWrap, &vicNodeId, &aggNodeId, &nomCouple,
        &minCouple, &maxCouple) != MWXSucces)
    {
        MWX_Printf((char*) "ERROR -- Failed to get CEdge data\n");
        continue;
    }
    MWX_Printf((char*) "%d  %d ---> %d (%g %g %g)\n", count++, vicNodeId,
```

```

        aggNodeId, nomCouple, minCouple, maxCouple);
}

if (MWXDb_Close_PvxRLCCoupleNetIteration( pvxDesign, &rlcCoupleNetWrap)
    != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to terminate RLCCoupleNet
        iteration\n");
    return (MWX_FALSE_SCM);
}

```

MWXDb_Get_PvxRLCCoupleNetAggNetInfo

Gets the RLCCoupleNet aggressor net information.

Syntax

```

int MWXDb_Get_PvxRLCCoupleNetAggNetInfo (
    void *pvxDesign, /* INPUT */
    MWXPvxObjData_t *vicNetWrap, /* INPUT */
    MWXPvxObjData_t *coupleNetWrap, /* INPUT */
    MWXPvxObjData_t *aggNetWrap /* OUTPUT */
)

```

Parameters

pvxDesign	Specifies the handle of the PARA design.
vicNetWrap	Specifies the record of the victim net.
coupleNetWrap	Specifies the RLCCoupleNet record.
aggNetWrap	Specifies the record of the aggressor net.

Return

MWXSuccess	Occurs if the RLCCoupleNet aggressor information is successfully returned.
MWXFail	Occurs if the RLCCoupleNet aggressor information is not returned.

Example

```

    MWXPvxObjData_t aggNetWrap;

    if (MWXDb_Get_PvxRLCCoupleNetAggNetInfo( pvxDesign, &rlcNetWrap,
        &rlcCoupleNetWrap, &aggNetWrap) != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to get aggressor net data\n");
        continue;
    }

    if (MWXDb_Get_PvxRLCNetInfo( pvxDesign, &aggNetWrap, &netName,

```

```

        &numOfNode, &numOfEdge) != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to get aggressor net name\n");
    return (MWX_FALSE_SCM);
}
MWX_Printf((char*) "Agg netName : %s\n", netName);

```

MWXDb_Get_PvxRLCCoupleNetInfo

Gets the RLCCoupleNet information.

Syntax

```

int MWXDb_Get_PvxRLCCoupleNetInfo (
    void *pvxDesign, /* INPUT */
    MWXPvxObjData_t *coupleNetWrap, /* INPUT */
    MWXPvxObjData_t *net1Wrap, /* OUTPUT */
    MWXPvxObjData_t *net2Wrap /* OUTPUT */
)

```

Parameters

pvxDesign	Specifies the handle of the PARA design.
coupleNetWrap	Specifies the record of the RLCCoupleNet.
net1Wrap	Specifies the record of net1.
net2Wrap	Specifies the record of net2.

Return

MWXSuccess	Occurs if the RLCCoupleNet information is successfully returned.
MWXFail	Occurs if the RLCCoupleNet information is not returned.

Example

```

MWXPvxObjData_t rlcNetWrap;
MWXPvxRLCNetField_t dataType;
MWXPvxObjData_t rlcCoupleNetWrap;
MWXPvxObjData_t net1Wrap, net2Wrap;

/* Couple Net */
dataType = MWXPvxRLCCoupleNetType;
if (MWXDb_Iter_PvxRLCNetMember_ByType( &rlcNetWrap, dataType)
    != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to start iteration for couple net
type\n");
    return (MWX_FALSE_SCM);
}

```



```

while ((rc = MWXDb_Iter_Next_PvxRLCNetMember_ByType(
    &rlcNetWrap, dataType,&rlcCoupleNetWrap)) != MWXIterDone) {
    if (rc == MWXFail) {
        MWX_Printf((char*)"ERROR -- Fail to Iterate PvxRLCNetMember by
type\n");
        continue;
    }
    if (MWXDb_Get_PvxRLCCoupleNetInfo( pvxDesign, &rlcCoupleNetWrap,
        &net1Wrap, &net2Wrap) != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to get net1/net2 data\n");
        continue;
    }

    if (MWXDb_Get_PvxRLCNetInfo( pvxDesign, &net1Wrap, &netName,
        &numOfNode, &numOfEdge) != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to get net1 information\n");
        return (MWX_FALSE_SCM);
    }
    MWX_Printf((char*) "Net 1:   netName=%s   numOfNode=%d   numOfEdge=%d \n",
        netName, numOfNode, numOfEdge);
}

```

MWXDb_Get_PvxRLCEdgeInfo

Gets the RLCEdge information.

Syntax

```

int MWXDb_Get_PvxRLCEdgeInfo (
    void *pvxDesign, /* INPUT */
    MWXPvxObjData_t *rlcEdgeWrap, /* INPUT */
    int *nodeId1, /* OUTPUT */
    int *nodeId2, /* OUTPUT */
    float *nomRes, /* OUTPUT */
    float *minRes, /* OUTPUT */
    float *maxRes /* OUTPUT */
)

```

Parameters

<code>pvxDesign</code>	Specifies the handle of the PARA design.
<code>rlcEdgeWrap</code>	Specifies the RLCEdge record.
<code>nodeId1</code>	Specifies the identifier of the node 1.
<code>nodeId2</code>	Specifies the identifier of the node 2.
<code>nomRes</code>	Specifies the value of nominal resistance.
<code>minRes</code>	Specifies the value of minimum resistance.
<code>maxRes</code>	Specifies the value of maximum resistance.

Return

<code>MWXSuccess</code>	Occurs if the RLCEdge information is successfully returned.
<code>MWXFail</code>	Occurs if the RLCEdge information is not returned.

Example

```
MWXPvxObjData_t rlcNetWrap;
MWXPvxRLCNetField_t dataType;
MWXPvxObjData_t rlcEdgeWrap;

dataType = MWXPvxRLCEdgeType;
if (MWXDb_Iter_PvxRLCNetMember_ByType( &rlcNetWrap, dataType)
    != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to start iteration for RLCEdge
type\n");
    return (MWX_FALSE_SCM);
}

count = 0;
while ((rc = MWXDb_Iter_Next_PvxRLCNetMember_ByType(
    &rlcNetWrap, dataType,&rlcEdgeWrap)) != MWXIterDone) {
    if (rc == MWXFail){
        MWX_Printf((char*)"ERROR -- Failed to iterate next
PvxRLCNetMember\n");
        continue;
    }
    if (MWXDb_Get_PvxRLCEdgeInfo( pvxDesign, &rlcEdgeWrap, &rlcNodeId1,
        &rlcNodeId2, &nomRes, &minRes, &maxRes) != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to get RLCEdge data\n");
        continue;
    }
    MWX_Printf((char*) "%d %d ---> %d (%g %g %g)\n", count++, rlcNodeId1,
        rlcNodeId2, nomRes, minRes, maxRes);
}
```

MWXdB_Get_PvxRLCNet_ByName

Gets the RLCNet handle for the given net name.

Syntax

```
int MWXdB_Get_PvxRLCNet_ByName (
    void *pvxDesign, /* INPUT */
    char *netName, /* INPUT */
    MWXPvxObjData_t *rlcNetWrap /* OUTPUT */
)
```

Parameters

pvxDesign	Specifies the handle of the parasitic design.
netName	Specifies the name of the net.
rlcNetWrap	Specifies the handle of RLCNet.

Return

MWXSucces	Occurs if the RLCNet handle is successfully returned.
MWXFail	Occurs if the RLCNet handle is not returned.

Example

```
MWXPvxObjData_t rlcNetWrap;
char strNetName[256];

if (MWXdB_Get_PvxRLCNet_ByName( pvxDesign, strNetName, &rlcNetWrap) !=
    MWXSucces) {
    MWX_Printf((char*) "ERROR -- Failed to get RLCNet handle for net %s\n",
        strNetName);
    return (MWX_FALSE_SCM);
}
```

MWXdB_Get_PvxRLCNetInfo

Gets the net data of the given RLCNet record.

Syntax

```
int MWXdB_Get_PvxRLCNetInfo (
    void *pvxDesign, /* INPUT */
    MWXPvxObjData_t *rlcNetWrap, /* INPUT */
    char **netName, /* OUTPUT */
    int32 *numOfNode, /* OUTPUT */
    int32 *numOfEdge /* OUTPUT */
)
```

MWXdB_Get_PvxRLCNet_ByName

Parameters

<code>pvxDesign</code>	Specifies the handle of the PARA design.
<code>rlcNetWrap</code>	Specifies the record of RLCNet.
<code>netName</code>	Specifies the name of the net to be returned.
<code>numOfNode</code>	Specifies the number of nodes.
<code>numOfEdge</code>	Specifies the number of edges.

Return

<code>MWXSuccess</code>	Occurs if the RLCNet information is successfully returned.
<code>MWXFail</code>	Occurs if the RLCNet information is not returned.

Example

```
MWXPvxObjData_t rlcNetWrap;
char *netName, strNetName[256];
int  numOfNode, numOfEdge;

if (MWXDb_Get_PvxRLCNet_ByName( pvxDesign, strNetName, &rlcNetWrap)
    != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to get RLCNet handle for net %s\n",
               strNetName);
    return (MWX_FALSE_SCM);
}

if (MWXDb_Get_PvxRLCNetInfo( pvxDesign, &rlcNetWrap, &netName, &numOfNode,
                             &numOfEdge) != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to get RLCNet information,\n");
    return (MWX_FALSE_SCM);
}

MWX_Printf((char*) "\nNet %s    numOfNode = %d    numOfEdge = %d\n", netName,
           numOfNode, numOfEdge);
```

MWXDb_Get_PvxRLCNodeInfo

Gets the RLCNode information.

Syntax

```
int MWXDb_Get_PvxRLCNodeInfo (
    void *pvxDesign, /* INPUT */
    MWXPvxObjData_t *rlcNodeWrap, /* INPUT */
    int32 rlcType, /* INPUT */
    int32 *nodeType, /* OUTPUT */
    ...)
```

```

        int32 *nodeId, /* OUTPUT */
        float *nomCap, /* OUTPUT */
        float *minCap, /* OUTPUT */
        float *maxCap /* OUTPUT */
    )

```

Parameters

<code>pvxDesign</code>	Specifies the handle of the PARA design.
<code>rlcNodeWrap</code>	Specifies the record of RLCNode.
<code>rlcType</code>	Specifies the type of tree or mesh.
<code>nodeType</code>	Specifies the type of node (1 means termNode, 0 means subNode).
<code>nodeId</code>	Specifies the identifier of the node.
<code>nomCap</code>	Specifies the value of nominal capacitance.
<code>minCap</code>	Specifies the value of minimum capacitance.
<code>maxCap</code>	Specifies the value of maximum capacitance.

Return

<code>MWXSucccess</code>	Occurs if the RLCNode information is successfully returned.
<code>MWXFail</code>	Occurs if the RLCNode information is not returned.

Example

```

MWXPvxObjData_t rlcNetWrap;
MWXPvxRLCNetField_t dataType;
MWXPvxObjData_t rlcNodeWrap;

/* Node */
dataType = MWXPvxRLCNodeType;
if (MWXDb_Iter_PvxRLCNetMember_ByType( &rlcNetWrap, dataType)
    != MWXSucccess) {
    MWX_Printf((char*) "ERROR -- Failed to start iteration for
        RLCNode type\n");
    return (MWX_FALSE_SCM);
}

rlcType = MWXcPvxRLCTree;
count = 0;
while ((rc = MWXDb_Iter_Next_PvxRLCNetMember_ByType(
    &rlcNetWrap, dataType,&rlcNodeWrap)) != MWXIterDone) {
    if(rc == MWXFail) {

```

```

        MWX_Printf((char*)"ERROR -- Failed to iterate next PvxRLCNetMember\n");
        continue;
    }
    if (MWXDb_Get_PvxRLCNodeInfo( pvxDesign, &rlcNodeWrap, rlcType, &nodeType,
        &rlcNodeId, &nomCap, &minCap, &maxCap) != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to get RLCNode data\n");
        continue;
    }
    MWX_Printf((char*) "%d    id:%d (%g %g %g)\n", count++, rlcNodeId,
        nomCap, minCap, maxCap);
}

```

MWXDb_Get_PvxRLCPortInfo

Gets the RLCPort data information where instance name will be NULL for boundary port.

Syntax

```

int MWXDb_Get_PvxRLCPortInfo (
    void *pvxDesign, /* INPUT */
    MWXPvxObjData_t *rlcPortWrap, /* INPUT */
    int32 *nodeId, /* OUTPUT */
    char **instName, /* OUTPUT */
    char **termName /* OUTPUT */
)

```

Parameter

pvxDesign	Specifies the handle of the PARA design.
rlcPortWrap	Specifies the record of RLCNet.
nodeId	Specifies the identifier of the node.
instName	Specifies the name of the cell instance.
termName	Specifies the name of the port.

Return

MWXSuccess	Occurs if the RLCPort information is successfully returned.
MWXFail	Occurs if the RLCPort information is not returned.

Example

```

MWXPvxObjData_t rlcNetWrap;
MWXPvxRLCNetField_t dataType;
MWXPvxObjData_t rlcPortWrap;

```

```

/* Port */
dataType = MWXPvxRLCPortType;
if (MWXDb_Iter_PvxRLCNetMember_ByType( &rlcNetWrap, dataType)
    != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to start iteration for
        RLCPort type\n");
    return (MWX_FALSE_SCM);
}

while ((rc = MWXDb_Iter_Next_PvxRLCNetMember_ByType( &rlcNetWrap, dataType,
    &rlcPortWrap)) != MWXIterDone) {
    if (rc == MWXFail){
        MWX_Printf((char*)"ERROR -- Failed to iterate next PvxRLCNetMember\n");
        continue;
    }
    if (MWXDb_Get_PvxRLCPortInfo( pvxDDesign, &rlcPortWrap, &portNodeId,
        &instName, &termName) != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to get RLCPort data\n");
        continue;
    }

    MWX_Printf((char*) " *I %s/%s    nodeId: %d\n", instName, termName,
        portNodeId);
}

```

MWXDb_Init_PvxDesign

Initializes the runtime parasitic data layer application for the given pvCellName. It opens the PARA cell and performs initial checks. It also opens the cell view if provided. You need to provide a cell view if the parasitic objects cross-reference with the object ID in cell view. The majority of the Milkyway database follows this referencing. The cell view is not required if the PARA view is self-contained, in other words, if it stores names instead of referencing with the cell view.

Syntax

```

int MWXDb_Init_PvxDesign (
    char *libraryName, /* INPUT */
    char *pvCellName, /* INPUT */
    char *cvCellName, /* INPUT */
    void **pvxDDesign /* OUTPUT */
)

```

Parameters

libraryName	Specifies the name of the library containing the PARA view and cell.
pvCellName	Specifies the name of the PARA cell to be opened.
cvCellName	Specifies the name of the cell to be opened.
pvxDesign	Specifies the handle of the parasitic design.

Return

MWXSuccess	Occurs if the PARA application is successfully initialized.
MWXFail	Occurs if the PARA application fails to initialize.

Example

```
void *pvxDesign = NULL;
if (MWXDb_Init_PvxDesign( libName, pvCellName, cvCellName, &pvxDesign)
    != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to initialize the
        para design\n");
    return (MWX_FALSE_SCM);
}
```

MWXDb_Init_PvxRLCCoupleNetIteration

Initializes the iteration for the RLCCoupleNet.

Syntax

```
int MWXDb_Init_PvxRLCNetIteration (
    void *pvxDesign, /* INPUT */
    MWXPvxObjData_t *coupleNetWrap /* INPUT/OUTPUT */
)
```


Parameters

<code>pvxDesign</code>	Specifies the handle of the PARA design.
<code>coupleNetWrap</code>	Specifies the record of RLCCoupleNet.

Return

<code>MWXSuccess</code>	Occurs if the RLCCoupleNet iteration successfully initializes.
<code>MWXFail</code>	Occurs if the RLCCoupleNet iteration fails to initialize.

Example

```
MWXPvxObjData_t rlcCoupleNetWrap;

while ((rc = MWXDb_Iter_Next_PvxRLCNetMember_ByType(
    &rlcNetWrap, dataType,&rlcCoupleNetWrap)) != MWXIterDone) {
    if(rc == MWXFail){
        MWX_Printf((char*)"Failed to iterate next PvxRLCNetMember\n");
        continue;
    }
    if (MWXDb_Init_PvxRLCCoupleNetIteration( pvxDesign,
&rlcCoupleNetWrap)
        != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to initialize RLCCoupleNet
            iteration\n");
        return (MWX_FALSE_SCM);
    }
    ... /* data processing */
}
```

MWXDb_Init_PvxRLCNetIteration

Initializes the iteration for the RLCNet.

Syntax

```
int MWXDb_Init_PvxRLCNetIteration (
    void *pvxDesign, /* INPUT */
    MWXPvxObjData_t *rlcNetWrap /* INPUT/OUTPUT */
)
```

Parameters

<code>pvxDesign</code>	Specifies the handle of the PARA design.
<code>rlcNetWrap</code>	Specifies the record of the RLCNet.

Return

<code>MWXSuccess</code>	Occurs if the RLCNet iteration is successfully initialized.
<code>MWXFail</code>	Occurs if the RLCNet iteration fails to initialize.

Example

```
if (MWXDb_Get_PvxRLCNet_ByName( pvxDesign, strNetName,
                                &rlcNetWrap)
    != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to get RLCNet handle for net %s\n",
               strNetName);
    return (MWX_FALSE_SCM);
}

if (MWXDb_Init_PvxRLCNetIteration( pvxDesign, &rlcNetWrap) != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to initialize RLCNet iteration\n");
    return (MWX_FALSE_SCM);
}
```

MWXDb_Iter_Next_PvxRLCCoupleNetMember_ByType

Traverses the RLCCoupleNet member data by type and returns the current element.

Syntax

```
int MWXDb_Iter_Next_PvxRLCCoupleNetMember_ByType (    MWXPvxObjData_t
*coupleNetWrap, /* INPUT/OUTPUT */    MWXPvxRLCCoupleNetField_t dataType,
/* INPUT */    MWXPvxObjData_t *memdata /* OUTPUT */)
```

Parameters

<code>coupleNetWrap</code>	Specifies the record of the RLCCoupleNet.
<code>dataType</code>	Specifies the type of RLCCoupleNet to be iterated.
<code>memData</code>	Specifies the record of member data to be returned.

Return

<code>MWXSuccess</code>	Occurs if the RLCCoupleNet's current member is successfully returned.
<code>MWXFail</code>	Occurs if the RLCCoupleNet's current member is not returned.
<code>MWXIterDone</code>	Occurs if the end of the object iteration is reached.

Example

```
MWXPvxObjData_t rlcCoupleNetWrap;
MWXPvxObjData_t cEdgeWrap;

if (MWXDb_Init_PvxRLCCoupleNetIteration( pvxDsgn, &rlcCoupleNetWrap)
    != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to initialize RLCCoupleNet
                    iteration\n");
    return (MWX_FALSE_SCM);
}

cDataType = MWXPvxCEdgeType;
while ((rc = MWXDb_Iter_Next_PvxRLCCoupleNetMember_ByType(
    &rlcCoupleNetWrap, cDataType, &cEdgeWrap)) != MWXIterDone) {
    if (rc == MWXFail){
        MWX_Printf((char*)
            "ERROR -- Failed to iterate next PvxRLCCoupleNetMember\n");
        continue;
    }
    ... /* data processing */
}

if (MWXDb_Close_PvxRLCCoupleNetIteration( pvxDsgn, &rlcCoupleNetWrap)
    != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to terminate RLCCoupleNet
                    iteration\n");
    return (MWX_FALSE_SCM);
}
```

MWXDb_Iter_Next_PvxRLCNetMember_ByType

Traverses the member data by type and returns the current element.

Syntax

```
int MWXDb_Iter_Next_PvxRLCNetMember_ByType (
    MWXPvxObjData_t *rlcNetWrap, /* INPUT/OUTPUT */
    MWXPvxRLCNetField_t dataType, /* INPUT */
    MWXPvxObjData_t *memData /* OUTPUT */
)
```

Parameters

rlcNetWrap	Specifies the record of the RLCNet.
dataType	Specifies the type of RLCNet being iterated.
memData	Specifies the record of member data to be returned.

Return

MWXSuccess	Occurs if the current element is successfully returned.
MWXFail	Occurs if the current element is not returned.
MWXIterDone	Occurs if the end of the object iteration is reached.

Example

```
MWXPvxObjData_t rlcNetWrap;
MWXPvxRLCNetField_t dataType;
MWXPvxObjData_t rlcPortWrap;

dataType = MWXPvxRLCPortType;
if (MWXDb_Iter_PvxRLCNetMember_ByType( &rlcNetWrap, dataType)
    != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to start iteration for RLCPort type\n");
    return (MWX_FALSE_SCM);
}

while ((rc = MWXDb_Iter_Next_PvxRLCNetMember_ByType( &rlcNetWrap, dataType,
    &rlcPortWrap)) != MWXIterDone) {
    if(rc == MWXFail){
        MWX_Printf((char*) "ERROR -- Failed to iterate next PvxRLCNetMember\n");
        continue;
    }
    ... /* data processing */
}
```

MWXDb_Iter_PvxRLCNetMember_ByType

Starts the iteration for RLCNet member by the data type specified.

Syntax

```
int MWXDb_Iter_PvxRLCNetMember_ByType (
    MWXPvxObjData_t *rlcNetWrap, /* INPUT/OUTPUT */
    MWXPvxRLCNetField_t dataType /* INPUT */
)
```

Parameters

rlcNetWrap	Specifies the record of the RLCNet.
dataType	Specifies the type of RLCNet to be iterated.

Return

MWXSuccess	Occurs if the RLCNet iteration is successfully started by the given data type.
MWXFail	Occurs if the RLCNet iteration fails to start.

Example

```
MWXPvxObjData_t rlcNetWrap;
MWXPvxRLCNetField_t dataType;

if (MWXDb_Init_PvxRLCNetIteration( pvxDsgn, &rlcNetWrap) != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to initialize RLCNet iteration\n");
    return (MWX_FALSE_SCM);
}

/* Port */
dataType = MWXPvxRLCPortType;
if (MWXDb_Iter_PvxRLCNetMember_ByType( &rlcNetWrap, dataType)
    != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to start iteration for RLCPort
                    type\n");
    return (MWX_FALSE_SCM);
}
```

Dynamic PARA View Pvx* Example

The following example uses Pvx* commands and shows the initialization of a design framework and the creation of a parasitic runtime data structure. After the RLC tree is constructed for the specified net, the corresponding data is obtained by traversing several object types. The example also shows the accessing of coupling mesh information. The design is then closed to free allocated memory.

```
#include <stdlib.h>
#include <malloc.h>
#include <assert.h>
#include "mw.h"
```

```

#ifdef __cplusplus
#include "mw_x_proto.h"
extern "C" {
    MWXSCM_t PvxDesignTest();
    void mwInitPvxDesignTest();
}
#endif

MWXSCM_t mwPvxDesignTest (MWXSCM_t list)
{
    void *pvxDesign = NULL;
    char libName[256], pvCellName[256], cvCellName[256];
    int rc;
    MWXPvxObjData_t rlcNetWrap;
    MWXPvxObjData_t rlcPortWrap, rlcNodeWrap, rlcEdgeWrap,
    rlcCoupleNetWrap;
    MWXPvxObjData_t aggNetWrap, cEdgeWrap, aggNodeWrap;
    MWXPvxRLCNetField_t dataType;
    MWXPvxRLCCoupleNetField_t cDataType;
    char strNetName[256];
    char *netName;
    int numOfNode, numOfEdge;
    int portNodeId;
    char *instName, *termName;
    int rlcType, nodeType, rlcNodeId;
    float nomCap, minCap, maxCap;
    int count;
    int rlcNodeId1, rlcNodeId2;
    float nomRes, minRes, maxRes;
    int vicNodeId, aggNodeId;
    float nomCouple, minCouple, maxCouple;
    MWXPvxObjData_t net1Wrap, net2Wrap;

    if (MWXSc_Scanf(list, (char*) "ssss", libName, pvCellName, cvCellName,
                    strNetName) != 4) {
        MWX_Printf((char*) "ERROR -- mwPvxDesignTest requires 4 arguments\n");
        return (MWX_FALSE_SCM);
    }

    if (MWXDb_Init_PvxDesign( libName, pvCellName, cvCellName, &pvxDesign)
        != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to initialize the para design\n");
        return (MWX_FALSE_SCM);
    }

    if (MWXDb_Create_PvxRCNetworkForAllNets( pvxDesign) != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to create RC network\n");
        return (MWX_FALSE_SCM);
    }

    if (MWXDb_Get_PvxRLCNet_ByName( pvxDesign, strNetName, &rlcNetWrap) !=
        MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to get RLCNet handle for net %s\n",
                    strNetName);
        return (MWX_FALSE_SCM);
    }
}

```

```

if (MWXDb_Get_PvxRLCNetInfo( pvxDsgn, &rlcNetWrap, &netName, &numOfNode,
                             &numOfEdge) != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to get RLCNet information\n");
    return (MWX_FALSE_SCM);
}

    MWX_Printf((char*) "\nNet %s    numOfNode = %d    numOfEdge = %d\n", netName,
               numOfNode, numOfEdge);

if (MWXDb_Init_PvxRLCNetIteration( pvxDsgn, &rlcNetWrap) != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to initialize RLCNet iteration\n");
    return (MWX_FALSE_SCM);
}

/* Port */
dataType = MWXPvxRLCPortType;
if (MWXDb_Iter_PvxRLCNetMember_ByType( &rlcNetWrap, dataType) != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to start iteration for RLCPort type\n");
    return (MWX_FALSE_SCM);
}
while ((rc = MWXDb_Iter_Next_PvxRLCNetMember_ByType( &rlcNetWrap, dataType,
                                                    &rlcPortWrap)) != MWXIterDone) {
    if (rc == MWXFail) {
        MWX_Printf((char*) "ERROR -- Failed to iterate next PvxRLCNetMember\n");
        continue;
    }
    if (MWXDb_Get_PvxRLCPortInfo( pvxDsgn, &rlcPortWrap, &portNodeId,
                                  &instName, &termName) != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to get RLCPort data\n");
        continue;
    }

    MWX_Printf((char*) "*I %s/%s    nodeId: %d\n", instName, termName,
               portNodeId);
}
/* End of Port */

/* Node */
dataType = MWXPvxRLCNodeType;
if (MWXDb_Iter_PvxRLCNetMember_ByType( &rlcNetWrap, dataType) != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to start iteration for RLCNode type\n");
    return (MWX_FALSE_SCM);
}
MWX_Printf((char*) "Cap section\n");
rlcType = MWXcPvxRLCTree;
count = 0;
while ((rc = MWXDb_Iter_Next_PvxRLCNetMember_ByType( &rlcNetWrap, dataType,
                                                    &rlcNodeWrap)) != MWXIterDone) {
    if (rc == MWXFail) {
        MWX_Printf((char*) "ERROR -- Failed to iterate next PvxRLCNetMember\n");
        continue;
    }
    if (MWXDb_Get_PvxRLCNodeInfo( pvxDsgn, &rlcNodeWrap, rlcType,
                                  &nodeType,
                                  &rlcNodeId, &nomCap, &minCap, &maxCap) != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to get RLCNode data\n");
        continue;
    }
}

```

```

        MWX_Printf((char*) "%d id:%d (%g %g %g)\n", count++, rlcNodeId,
                    nomCap, minCap, maxCap);
    }
    /* End of Node */

    /* Edge */
    dataType = MWXPvxRLCEdgeType;
    if (MWXDb_Iter_PvxRLCNetMember_ByType( &rlcNetWrap, dataType) != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to start iteration for RLCEdge type\n");
        return (MWX_FALSE_SCM);
    }
    MWX_Printf((char*) "Res section\n");
    count = 0;
    while ((rc = MWXDb_Iter_Next_PvxRLCNetMember_ByType( &rlcNetWrap, dataType,
        &rlcEdgeWrap)) != MWXIterDone) {
        if (rc == MWXFail) {
            MWX_Printf((char*) "ERROR -- Failed to iterate next PvxRLCNetMember\n");
            continue;
        }
        if (MWXDb_Get_PvxRLCEdgeInfo( pvxDesign, &rlcEdgeWrap, &rlcNodeId1,
            &rlcNodeId2, &nomRes, &minRes, &maxRes) != MWXSuccess) {
            MWX_Printf((char*) "ERROR -- Failed to get RLCEdge data\n");
            continue;
        }
        MWX_Printf((char*) "%d %d ---> %d (%g %g %g)\n", count++, rlcNodeId1,
            rlcNodeId2, nomRes, minRes, maxRes);
    }
    /* End of Edge */

    /* Couple Net */
    dataType = MWXPvxRLCCoupleNetType;
    if (MWXDb_Iter_PvxRLCNetMember_ByType( &rlcNetWrap, dataType) != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to start iteration for couple net
            type\n");
        return (MWX_FALSE_SCM);
    }
    MWX_Printf((char*) "Couple section\n");
    count = 0;
    while ((rc = MWXDb_Iter_Next_PvxRLCNetMember_ByType( &rlcNetWrap, dataType,
        &rlcCoupleNetWrap)) != MWXIterDone) {
        if (rc == MWXFail) {
            MWX_Printf((char*) "ERROR -- Failed to iterate next PvxRLCNetMember\n");
            continue;
        }
        if (MWXDb_Get_PvxRLCCoupleNetInfo( pvxDesign, &rlcCoupleNetWrap,
            &net1Wrap, &net2Wrap) != MWXSuccess) {
            MWX_Printf((char*) "ERROR -- Failed to get net1/net2 data\n");
        }

        if (MWXDb_Get_PvxRLCNetInfo( pvxDesign, &net1Wrap, &netName,
            &numOfNode, &numOfEdge) != MWXSuccess) {
            MWX_Printf((char*) "ERROR -- Failed to get net1 name\n");
            return (MWX_FALSE_SCM);
        }
        MWX_Printf((char*) "Net 1: netName=%s numOfNode=%d numOfEdge=%d \n",
            netName, numOfNode, numOfEdge);

        if (MWXDb_Get_PvxRLCNetInfo( pvxDesign, &net2Wrap, &netName,

```



```

        &numOfNode, &numOfEdge) != MWXSuccess) {
            MWX_Printf((char*) "ERROR -- Failed to get net2 name\n");
            return (MWX_FALSE_SCM);
        }
        MWX_Printf((char*) "Net 2:  netName=%s    numOfNode=%d    numOfEdge=%d \n",
                    netName, numOfNode, numOfEdge);

    if (MWXDb_Get_PvxRLCCoupleNetAggNetInfo( pvxDesign, &rlcNetWrap,
        &rlcCoupleNetWrap, &aggNetWrap) != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to get aggressor net data\n");
        continue;
    }

    if (MWXDb_Get_PvxRLCNetInfo( pvxDesign, &aggNetWrap, &netName,
        &numOfNode, &numOfEdge) != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to get aggressor net name\n");
        return (MWX_FALSE_SCM);
    }
    MWX_Printf((char*) "Agg netName : %s\n", netName);

    if (MWXDb_Init_PvxRLCCoupleNetIteration( pvxDesign, &rlcCoupleNetWrap)
        != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to initialize RLCCoupleNet
            iteration\n");
        return (MWX_FALSE_SCM);
    }

    cDataType = MWXPvxCEdgeType;
    while ((rc = MWXDb_Iter_Next_PvxRLCCoupleNetMember_ByType(
        &rlcCoupleNetWrap, cDataType, &cEdgeWrap)) != MWXIterDone) {
        if (rc == MWXFail) {
            MWX_Printf((char*) "ERROR -- Failed to iterate next
                PvxRLCCoupleNetMember\n");
            continue;
        }
        if (MWXDb_Get_PvxCEdgeInfo( pvxDesign, &cEdgeWrap, &rlcNetWrap,
            &aggNodeWrap, &vicNodeId, &aggNodeId, &nomCouple, &minCouple,
            &maxCouple) != MWXSuccess) {
            MWX_Printf((char*) "ERROR -- Failed to get CEEdge data\n");
            continue;
        }
        MWX_Printf((char*) "%d  %d ---> %d (%g %g %g)\n", count++, vicNodeId,
            aggNodeId, nomCouple, minCouple, maxCouple);
    }

    if (MWXDb_Close_PvxRLCCoupleNetIteration( pvxDesign,
        &rlcCoupleNetWrap)
        != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to terminate RLCCoupleNet
            iteration\n");
        return (MWX_FALSE_SCM);
    }
}
/* End of Couple Net */

if (MWXDb_Close_PvxRLCNetIteration( pvxDesign, &rlcNetWrap) != MWXSuccess) {
    MWX_Printf((char*) "ERROR -- Failed to terminate RLCNet iteration\n");
    return (MWX_FALSE_SCM);
}

```

```

    }

    if (MWXDb_Close_PvxDesign( pvxDesign) != MWXSuccess) {
        MWX_Printf((char*) "ERROR -- Failed to close the para design\n");
        return (MWX_FALSE_SCM);
    }

    return (MWX_TRUE_SCM);
}

/* Define a function to register the new command. */
void mwInitPvxDesignTest()
{
    MWXSc_Register_Function ((char*) "mwPvxDesignTest", mwPvxDesignTest);
}

```

Index

A

accessDirection field 3-106
accessing libraries 2-2, 3-2
API methodology, understanding 1-8
AvantAr 1-12

B

BaseArray object 3-9
blockage area 2-5
Boundary object 3-11
BPVContactArray object 3-12

C

CEdge object 3-12
cell access 2-7
cell ID 2-7, 3-2
cell naming conventions 2-6
cell object functions, table of 2-8
CellEquivalentClass object 3-13
cellFlag field 3-106
CellInfo object 3-14
CellInst object 3-16
CellInstArray object 3-18
CellInstMaster object 3-21

- CellRegion object 3-23
- CellRow object 3-24
- classification of functions 1-10
- CLF file 2-2
- CLFRecord object 3-24
- ClkOptData object 3-25
- common object fields
 - accessDirection 3-106
 - cellFlag 3-106
 - recType 3-108
 - routeType 3-108
 - statusFlag 3-109
 - tableType 3-110
- Contact object 3-26
- ContactArray object 3-28

D

- data fields, getting 3-4
- database access mechanisms 1-5
- database object
 - BaseArray 3-9
 - Boundary 3-11
 - BPVContactArray 3-12
 - CEdge 3-12
 - CellEquivalentClass 3-13
 - CellInfo 3-14
 - CellInst 3-16
 - CellInstArray 3-18
 - CellInstMaster 3-21
 - CellRegion 3-23
 - CellRow 3-24
 - CLFRecord 3-24
 - ClkOptData 3-25
 - Contact 3-26
 - ContactArray 3-28
 - DelayTable 3-29
 - ErrorComponent 3-30
 - ErrorObj 3-31
 - ErrorType 3-33
 - ExpCellInst 3-34

Chapter IN:

IN-2

ExpPinInst 3-35
ExpPortInst 3-36
Glink 3-41
Group 3-42
HorizontalWire 3-47
IGRPath 3-48
InstPower 3-48
LEdge 3-49
LibInfo 3-49
Net 3-50
Path 3-54
Pin 3-55
PinAccessEdge 3-59
PinInstMaster 3-60
PlanGroup 3-62
Polygon 3-64
Port 3-65
PortFunc 3-67
PortInst 3-68
PortInstMaster 3-69
PortToPortDelay 3-70
PortToPortTable 3-71
PreRouteSegment 3-50
Rectangle 3-73
RectArray 3-74
REdge 3-74
RegionBlockage 3-75
RLCCoupleNet 3-75
RLCNet 3-77
RLCNode 3-79
RLCPort 3-79
RouteGuide 3-80
Text 3-97
TilePattern 3-98
VerticalWire 3-99
ViaRegion 3-100
WireDirection 3-102
WireMaster 3-102
WireTrack 3-104
database objects, working with 3-2
db-foreach command 3-3
dbOpenCel Scheme command 1-29

dbOpenCell 1-27
defining symbols 1-32
DelayTable object 3-29
direction field 3-105

E

ErrorComponent object 3-30
ErrorObj object 3-31
ErrorType object 3-33
ExpCellInst object 3-34
ExpPinInst object 3-35
ExpPortInst object 3-36
ExtraPin 3-36
ExtraPinInstMaster 3-37
ExtraPort 3-38
ExtraPortInst 3-39
ExtraPortInstMaster 3-40

F

Field accessor function prototype 3-9
function return codes 1-33

G

gap fill cell 2-5
Glink object 3-41
Group object 3-42

H

header files 1-28
HierCellInst 3-42
HierCellInstMaster 3-43
HierNet 3-45
HierPortInst 3-44
HierPortVectorInfo 3-46
HorizontalWire object 3-47

Chapter IN:

IN-4

I

IGRPath object 3-48

InstPower object 3-48

L

layout cell 2-5

LEdge object 3-49

LibInfo object 3-49

libOMWX.a 1-13

library 2-2

library ID 2-3

M

MMWXDb_Create_Group 4-67

MWX_Log 6-2

MWX_Printf 6-2

mxw_proto.h 1-12

MWXDb_Abort_Cell 4-2

MWXDb_Add_CellInst_ToPlanGroup_ByName 4-4

MWXDb_Add_CellInstance_ToCellRegion 4-3

MWXDb_Add_InstGroup_ToScanChain 8-2

MWXDb_Add_NetSubType 4-5

MWXDb_Add_NetSubType_ByName 4-7

MWXDb_Add_Object_ToGroup 4-8

MWXDb_Add_PartialScan_ToScanChain 8-3

MWXDb_Add_RefLib 4-9

MWXDb_Archive_CellToLib 4-11

MWXDb_Archive_HierDesign 4-12

MWXDb_Assign_VarRouteRule_ById 8-4

MWXDb_Assign_VarRouteRule_ByNetName 8-5

MWXDb_Attach_ChildCellInstToHierCellInst 8-7

MWXDb_Attach_ChildHierCellInstToHierCellInst 8-8

MWXDb_Attach_HierPortInstMasterToHierCellInstMaster 8-9

MWXDb_Attach_HierPortInstToHierCellInst 8-10

MWXDb_Attach_HierPortInstToHierNet 8-11

MWXDb_Attach_PortInstToHierNet 8-13

MWXdB_Begin_ReadLib 4-13
MWXdB_Begin_WriteLib 4-15
MWXdB_BooleanAnd_Objects 6-3
MWXdB_BooleanNot_Objects 6-4
MWXdB_BooleanOr_Objects 6-5
MWXdB_BooleanXor_Objects 6-6
MWXdB_Change_Cell_AccessMode 4-15
MWXdB_Check_Cell_Exist 4-16
MWXdB_Check_Cell_ExistInLibrary 4-17
MWXdB_Check_IdenticalCell 4-19
MWXdB_Check_RefLibControl 4-19
MWXdB_Cleanup_ShapeStruct 4-20
MWXdB_Clear_CellEEQClass 8-14
MWXdB_Clear_CellEQClass 8-15
MWXdB_Clear_CellLEQClass 8-16
MWXdB_Clear_CellMustJoinPin 4-21
MWXdB_Clear_RefLibControl 4-22
MWXdB_Close_Cell 4-23
MWXdB_Close_Cell_And_NoSave 4-25
MWXdB_Close_Cell_And_Save 4-26
MWXdB_Close_Lib 4-28
MWXdB_Close_PvxDesign 9-16
MWXdB_Close_PvxRLCCoupleNetIteration 9-17
MWXdB_Close_PvxRLCNetIteration 9-17
MWXdB_Connect 4-29
MWXdB_Connect_PortInstance_ByName 4-30
MWXdB_Convert_PathToPolygon 4-33
MWXdB_Convert_PathToRectangles 4-35
MWXdB_Convert_Point_ToDBPoint 4-31, 4-32, 4-37
MWXdB_Convert_PolygonToRectangles 4-38
MWXdB_Convert_RectangleToPolygon 4-39
MWXdB_Copy_Cell 4-41
MWXdB_Copy_CellFromAnotherLib 4-43
MWXdB_Copy_CellToAnotherLibDisk 4-45
MWXdB_Copy_CellWithTIM 4-46
MWXdB_Copy_Lib 4-47
MWXdB_Create_BaseArray 8-17
MWXdB_Create_BaseArrayRow 8-18

Chapter IN:

MWXDb_Create_Cell 4-49
MWXDb_Create_Cell_Property 4-50
MWXDb_Create_CellAttachFile 4-52
MWXDb_Create_CellBoundary 4-53
MWXDb_Create_CellEEQClass 8-19
MWXDb_Create_CellInst_IgnoreECO 8-20
MWXDb_Create_CellInst_WithoutCheckStatus 4-56
MWXDb_Create_CellInstance 4-54
MWXDb_Create_CellLEQClass 8-21
MWXDb_Create_CellRegion 4-58
MWXDb_Create_CellRow 4-59
MWXDb_Create_Contact_WithRoutType 8-23
MWXDb_Create_ContactArray_WithRoutType 8-24
MWXDb_Create_ContactCode 4-61
MWXDb_Create_CoreBaseArray 8-27
MWXDb_Create_ErrorComponents 4-62
MWXDb_Create_ErrorOb 4-64
MWXDb_Create_ErrorType 4-66
MWXDb_Create_Glink 8-28
MWXDb_Create_HierCellinst 8-29
MWXDb_Create_HierCellInstMaster 8-31
MWXDb_Create_HierNet 8-32
MWXDb_Create_HierPortInst 8-33
MWXDb_Create_HierPortInstMaster 8-36
MWXDb_Create_HierPortVectorInfo 8-37
MWXDb_Create_Horizontal_Wire 4-69
MWXDb_Create_Lib 4-70
MWXDb_Create_LibAttachFile 4-72
MWXDb_Create_Net 4-73
MWXDb_Create_ParameterSet 4-74
MWXDb_Create_Path 4-75
MWXDb_Create_PcellDefinition 8-39
MWXDb_Create_PcellInstance 8-40
MWXDb_Create_Pin 4-78
MWXDb_Create_PlanGroup 4-80, 4-82
MWXDb_Create_Polygon 4-84
MWXDb_Create_Port 4-86
MWXDb_Create_Property 4-88

MWXdB_Create_PvxRCNetworkForAllNets 9-18
MWXdB_Create_Rectangle 4-90
MWXdB_Create_Row 8-41
MWXdB_Create_ScanChain 8-43
MWXdB_Create_SGEdge 8-43
MWXdB_Create_SGPrimaryNode 8-44
MWXdB_Create_TdfPad 4-92
MWXdB_Create_Text 4-95
MWXdB_Create_TilePattern 8-46
MWXdB_Create_Track 8-47
MWXdB_Create_Vertical_Wire 8-50
MWXdB_Create_Wire_Master 4-97
MWXdB_Create_WireDirection 8-51
MWXdB_Define_VarRouteRule_Contact 8-53
MWXdB_Define_VarRouteRule_Spacing 8-54
MWXdB_Define_VarRouteRule_TaperLevel 8-56
MWXdB_Define_VarRouteRule_Width 8-57
MWXdB_Delete_AllRouteGuides 8-59
MWXdB_Delete_AllVarRouteRules 8-59
MWXdB_Delete_Cell 4-99
MWXdB_Delete_CellAttachFile 4-99
MWXdB_Delete_CellInstance_FromCellRegion 8-60
MWXdB_Delete_CellProperty_ByName 4-100
MWXdB_Delete_LibAttachFile 4-101
MWXdB_Delete_Object 4-102
MWXdB_Delete_ObjectArray 4-103
MWXdB_Delete_Objects_ByLayer 4-104
MWXdB_Delete_PcellDefinition 8-61
MWXdB_Delete_Pin 4-105
MWXdB_Delete_Property 4-106
MWXdB_Delete_Property_ByName 4-107
MWXdB_Delete_RefLib_ByName 4-108
MWXdB_Delete_ScanChain 8-62
MWXdB_Disconnect 4-109
MWXdB_Disconnect_PortInstance_ByName 4-110
MWXdB_Display_DiodePinInstMaster_FromDiodePortInstMaster 4-111
MWXdB_Display_DiodePortInst 4-112
MWXdB_Display_DiodePortInst_FromCellInst 4-113

Chapter IN:

MWXdB_Display_DiodePortInst_FromNet 4-114
MWXdB_Display_DiodePortInstMaster 4-115
MWXdB_Display_DiodePortInstMaster_FromDiodeCellInstMaster 4-116
MWXdB_Dump_AllVarRouteRules 8-63
MWXdB_Dump_TechFile 4-117
MWXdB_End_WriteLib 4-120
MWXdB_EndReadLib 4-119
MWXdB_Fetch_Object 4-122
MWXdB_Get_* 3-4
MWXdB_Get_AccessEdgeFromPin 4-123
MWXdB_Get_AllCellProperties 4-124, 8-64
MWXdB_Get_AllNetVarRouteRules 8-65
MWXdB_Get_AllObjectProperties 4-125
MWXdB_Get_AllPin_ByName 4-126
MWXdB_Get_AllRouteGuides 8-67
MWXdB_Get_AllVarRouteRules 8-68
MWXdB_Get_CaseSensitive 4-127
MWXdB_Get_Cell_EEQClass 8-69
MWXdB_Get_Cell_LEQClass 8-70
MWXdB_Get_CellAttachedFiles 4-129
MWXdB_Get_CellBBox 4-128, 4-131
MWXdB_Get_CellBoundary 4-132
MWXdB_Get_CellFullName 4-133
MWXdB_Get_CellFullPath 4-135
MWXdB_Get_CellInfo_cellNameAndView 4-136
MWXdB_Get_CellInfo_fullName 4-137
MWXdB_Get_CellInst_PowerDelayScale 8-71
MWXdB_Get_CellInst_PowerId 8-72
MWXdB_Get_CellInst_PowerSupplyVoltage 8-73
MWXdB_Get_CellInst_PowerTemperature 8-74
MWXdB_Get_CellInst_PowerValue 8-75
MWXdB_Get_CellInstance_ByName 4-137
MWXdB_Get_CellInstanceBBox 4-138
MWXdB_Get_CellInstanceOrArray_ByName 4-139
MWXdB_Get_CellInstCellType_Bit 4-142
MWXdB_Get_CellInstFromCellInstMaster 4-143
MWXdB_Get_CellLatestVersion 4-144
MWXdB_Get_CellName_By_CellId 4-145

MWXdB_Get_CellNames_ByView 4-147
MWXdB_Get_CellProperty_ByName 4-148
MWXdB_Get_CellRegionIdFromCellInstId 8-76
MWXdB_Get_ColorIDByName 4-151
MWXdB_Get_ColorName 5-2
MWXdB_Get_ColorRGB 5-2
MWXdB_Get_ColorRGBDefined 5-3
MWXdB_Get_ConnectedObject_ById 4-152
MWXdB_Get_Constants 4-153, 4-158
MWXdB_Get_ContactCode_ByName 4-154
MWXdB_Get_ContactCodeIdx_ByName 4-156
MWXdB_Get_ContactCodeNames 4-157
MWXdB_Get_ContactName_ByIndex 4-160
MWXdB_Get_CurrentLibId 4-161, 4-162
MWXdB_Get_DefaultContactByMetalLayers 4-163
MWXdB_Get_DefaultContactByViaLayer 4-164
MWXdB_Get_DesignRuleDiffNetMinSpacing 5-4
MWXdB_Get_DesignRuleMinEnclosure 5-5
MWXdB_Get_DesignRuleMinSpacing 5-6
MWXdB_Get_DesignStackable 5-7
MWXdB_Get_EditCell 4-165
MWXdB_Get_FatContactByMaskName 4-166
MWXdB_Get_GlobalNet_ByNameMap 8-78
MWXdB_Get_Group_ByName 4-167
MWXdB_Get_GroupMember 4-169
MWXdB_Get_HierRefLibMode 4-170
MWXdB_Get_InputData 4-170
MWXdB_Get_Instances_ByBoundingBoxSearch 4-171
MWXdB_Get_Layer* functions 5-1
MWXdB_Get_LayerBorderStyleName 5-9
MWXdB_Get_LayerCapacitance 5-10
MWXdB_Get_LayerColorName 5-8, 5-13
MWXdB_Get_LayerDisplayAttr 5-14
MWXdB_Get_LayerEtchFactor 5-15
MWXdB_Get_LayerFatWireRules 5-17
MWXdB_Get_LayerFringeCap 5-18
MWXdB_Get_LayerHeightFromSub 5-19
MWXdB_Get_LayerId_ByName 4-150, 4-172

Chapter IN:

MWXDb_Get_LayerId_ByNumber 4-173
MWXDb_Get_LayerInductance 5-20
MWXDb_Get_LayerLayoutAttr 5-21
MWXDb_Get_LayerLineStyleName 5-22
MWXDb_Get_LayerName 4-174
MWXDb_Get_LayerName_FromNumber 4-188, 5-23
MWXDb_Get_LayerNames 4-175
MWXDb_Get_LayerNumber_FromName 5-24
MWXDb_Get_LayerPhysicalAttr 5-25
MWXDb_Get_LayerResistance 5-26
MWXDb_Get_LayerStippleName 5-27
MWXDb_Get_LayerStubRouting 5-28
MWXDb_Get_LayerThickness 5-29
MWXDb_Get_LibAttachedFiles 4-177
MWXDb_Get_LibHierDelimiter 4-181
MWXDb_Get_LibId_ByCellId 4-178
MWXDb_Get_LibId_ByLibName 4-179
MWXDb_Get_LibName_ById 4-182
MWXDb_Get_LibViewNames 4-184
MWXDb_Get_LogicModelSubType 4-185
MWXDb_Get_MaskNames 4-186
MWXDb_Get_Net_ByName 4-188, 4-195
MWXDb_Get_NetLoad 8-79
MWXDb_Get_NetPort 4-190
MWXDb_Get_NetPortInst 4-191
MWXDb_Get_NetPowerId 8-80
MWXDb_Get_NetSwitchActivity 8-81
MWXDb_Get_NetWiring 4-192
MWXDb_Get_Objects_ByBBox 4-193
MWXDb_Get_Objects_ByType 4-196
MWXDb_Get_Objects_ByTypeAndLayer 4-198
MWXDb_Get_ObjectsCount 4-199
MWXDb_Get_ObjectsType 4-200
MWXDb_Get_OwnerLibId_BYCellName 4-200
MWXDb_Get_Pin_ByName 4-203
MWXDb_Get_PinFromPort 4-204
MWXDb_Get_PinInstMaster_numberOfPoint 4-205
MWXDb_Get_PinNet_ByName 4-205

MWXdB_Get_PlanGroup_ByName 4-206
MWXdB_Get_Port_ByName 4-207
MWXdB_Get_Port_name 3-4
MWXdB_Get_PortInstFromCellInst 4-208
MWXdB_Get_PrBoundary 4-209
MWXdB_Get_Property 4-210
MWXdB_Get_Property_ByName 4-212
MWXdB_Get_PvxCEdgeInfo 9-19
MWXdB_Get_PvxRLCCoupleNetAggNetInfo 9-21
MWXdB_Get_PvxRLCCoupleNetInfo 9-22
MWXdB_Get_PvxRLCEdgeInfo 9-23
MWXdB_Get_PvxRLCNet_ByName 9-25
MWXdB_Get_PvxRLCNetInfo 9-25
MWXdB_Get_PvxRLCNodeInfo 9-26
MWXdB_Get_PvxRLCPortInfo 9-28
MWXdB_Get_RefLib 4-214
MWXdB_Get_RefLibPathNames 4-215
MWXdB_Get_RegionPort_ByName 4-218
MWXdB_Get_RegionPort_Direction 4-218
MWXdB_Get_RouteGuide 8-82
MWXdB_Get_RouteGuide_PropertyValueArray 8-83
MWXdB_Get_SelectedSet 4-219
MWXdB_Get_TechCapacitance 5-30
MWXdB_Get_TechCurrent 5-33
MWXdB_Get_TechDielectric 5-34
MWXdB_Get_TechInductance 5-35
MWXdB_Get_TechName 5-36
MWXdB_Get_TechPower 5-37
MWXdB_Get_TechResistance 5-38
MWXdB_Get_TechResolution 5-39
MWXdB_Get_TechRowAbutable 5-40
MWXdB_Get_TechRowSpacing 5-41
MWXdB_Get_TechTime 5-42
MWXdB_Get_TechVoltage 5-43
MWXdB_Get_VarRouteRuleName_ByNetId 8-86
MWXdB_Get_VarRouteRuleSpacing_ByName 8-87
MWXdB_Get_VarRouteRuleTaperLevel_ByName 8-89
MWXdB_Get_VarRouteRuleWidth_ByName 8-91

Chapter IN:

IN-12

MWXDb_Get_VarVarRouteRuleContact_ByName 8-93
MWXDb_Get_ViaRegionFromPort 4-221
MWXDb_Get_WireDirection_layer 4-222
MWXDb_Get_WireDirection_preferedDirection 4-223
MWXDb_Get_WireMasterId 4-223
MWXDb_Get_WireTrack_bbox 4-224
MWXDb_Get_WireTrack_layer 4-225
MWXDb_Get_WireTrack_startTrack 4-226
MWXDb_Get_WireTrack_times 4-227
MWXDb_Get_WireTrack_trackDirection 4-228
MWXDb_Get_WireTrack_trackStep 4-229
MWXDb_Get_WireTrack_type 4-230
MWXDb_Init_PvxDesign 9-29
MWXDb_Init_PvxRLCCoupleNetIteration 9-30
MWXDb_Init_PvxRLCNetIteration 9-31
mwxdb_is_casesensitive 4-231
MWXDb_Is_NetSubTypeCoreAndPad 4-233
MWXDb_Is_NetSubTypeCoreOnly 4-233
MWXDb_Is_NetSubTypeCTSNet 4-234
MWXDb_Is_NetSubTypeCTSRootNet 4-235
MWXDb_Is_NetSubTypeNonCTSNet 4-236
MWXDb_Is_NetSubTypePadOnly 4-237
MWXDb_Is_Object_InGroup 4-238
MWXDb_Iter_LibCellViewPair 4-243
MWXDb_Iter_Next 4-244
MWXDb_Iter_Next_ExpPinInst 8-94
MWXDb_Iter_Next_LibCellViewPair 4-245
MWXDb_Iter_Next_PvxRLCCoupleNetMember_ByType 9-32
MWXDb_Iter_Next_PvxRLCNetMember_ByType 9-33
MWXDb_Iter_ObjectByType 4-248
MWXDb_Iter_PortInstToExpPinInst 8-95
MWXDb_Iter_PortToTiming 8-99
MWXDb_Iter_PvxRLCNetMember_ByType 9-34
MWXDb_Iter_Quit 4-250
MWXDb_Iter_Quit_LibCellViewPair 4-251
MWXDb_LoadDesignSetupDone 8-101
MWXDb_Merge_Objects 4-251
MWXDb_omwxInit 4-252

MWXdB_Open_Cell 1-27, 4-253
MWXdB_Open_Cell_ByLibId 4-255
MWXdB_Open_CellAttachFile 4-254
MWXdB_Open_Lib 4-258
MWXdB_Open_LibAttachFile 4-260, 4-263
MWXdB_Open_Library_ByAccessMode 4-261
MWXdB_Output_ClickedPoint 4-264
MWXdB_Place_CellInstanceInLegalLoc 4-265
MWXdB_Purge_AllPlanGroup 4-267
MWXdB_Purge_AllSingleNet 4-267
MWXdB_Purge_AllTdfPads 4-268
MWXdB_Purge_CellInstance_ByName 4-269
MWXdB_Purge_CellInstMaster 4-270
MWXdB_Purge_Group_ByName 4-271
MWXdB_Purge_Net 4-272
MWXdB_Purge_SingleObjectType 4-273
MWXdB_Rebuild_CellWithVersion 4-275
MWXdB_Rebuild_Library 4-276
MWXdB_Rebuild_Library_WithAccessCheck 4-278
mwxdb_recompute_cellbbox 4-280
MWXdB_Remove_InstGroup_FromSchanChain 8-103
MWXdB_Remove_Object_FromGroup 4-281
MWXdB_Rename_Cell 4-282
MWXdB_Rename_Lib 4-283
MWXdB_Replace_CellInst 4-285
MWXdB_Replace_CellTextChar 4-286
MWXdB_Replace_CellTextString 4-287
MWXdB_Replace_ChildCell 4-288
MWXdB_Replace_RefLib_ByName 4-289
MWXdB_Replace_TechInfo 4-290
MWXdB_Replace_TextChar 4-291
MWXdB_Replace_TextString 4-292
MWXdB_Reset_CellInstance_Traceable 8-104
MWXdB_Reset_Net_Traceable 8-104
MWXdB_Restore_HierDesign 8-105
MWXdB_Restore_Pin 4-293
MWXdB_RollBack_Cell 4-295
MWXdB_Save_Cell 4-295, 8-109

MWXdB_Set_CaseSensitive 4-296
MWXdB_Set_CellInst_PowerDelayScale 8-106
MWXdB_Set_CellInst_PowerSupplyVoltage 8-107
MWXdB_Set_CellInst_PowerTemperature 8-108
MWXdB_Set_CellInstance_Traceable 8-111
MWXdB_Set_CellInstanceECOStatus_ByName 8-110
MWXdB_Set_CellInstanceFixStatus_ByName 4-297
MWXdB_Set_CellInstancePlacement_ByName 4-298
MWXdB_Set_CellLayoutSubType 4-300
MWXdB_Set_CellLModelSubType 4-301
MWXdB_Set_CellMustJoin2Pin 8-112
MWXdB_Set_CellMustJoinPinPattern 8-113
MWXdB_Set_CellType 4-301
MWXdB_Set_CellType_Bit 4-302
MWXdB_Set_ErrorObjStatus 4-303
MWXdB_Set_ErrorTypeStatus 4-304
MWXdB_Set_GetBBoxCallBack 4-305
MWXdB_Set_GetLineCallBack 4-307
MWXdB_Set_GetPointCallBack 4-309
MWXdB_Set_HierRefLibMode 4-311
MWXdB_Set_HorizontalWire_routeType 8-114
MWXdB_Set_IOPad 4-312
MWXdB_Set_Net_Traceable 8-115
MWXdB_Set_Netload 8-116
MWXdB_Set_NetSubType 4-313
MWXdB_Set_NetSubType_ByName 4-314
MWXdB_Set_NetSwitchActivity 8-117
MWXdB_Set_NetType 4-315
MWXdB_Set_NetType_ByName 4-316
MWXdB_Set_RefControlFileMode 4-317
MWXdB_Set_RefLibControl 4-318
MWXdB_Set_ScanChain_DefaultPriority 8-118
MWXdB_Set_StdCell 4-319
MWXdB_Set_TdfLoadCapacitance 4-320
MWXdB_Set_TdfLoadCapacitanceInDouble 4-321
MWXdB_Set_TdfSrcCapacitanceList 4-321
MWXdB_Set_TdfSrcCapacitanceListInDouble 4-322
MWXdB_Set_TdfSrcResistanceList 4-323

MWXDb_Set_TdfSrcResistanceListInDouble 4-324
MWXDb_Set_TechFileName 4-325
MWXDb_Set_TopHierCellInst 8-119
MWXDb_Set_VerticalWire_routeType 8-120
MWXDb_Show_DiodePortConn 4-326
MWXDb_Startup_ShapeStruct 4-327
MWXDb_Store_CellAttachFile 4-328
MWXDb_Store_LibAttachFile 4-329
MWXDb_Transform_Object 4-330
MWXDb_Translate_MilkywayName_ToVerilogName 8-121
MWXDb_Translate_VerilogName_ToMilkywayName 8-122
MWXDb_Truncate_CellText 4-331
MWXDb_Truncate_Text 4-332
MWXDb_Undelete_Object 8-124
MWXDb_Update_CellProperty_ByName 4-333
MWXDb_Update_Property 4-335
MWXDb_Update_Property_ByName 4-337
mxw.h 1-12
MWXSc_* functions 7-1
MWXSc_Car 7-3
MWXSc_Cdr 7-4
MWXSc_Evaluate 7-4
MWXSc_Get_BooleanSCM 7-5
MWXSc_Get_DBIdSCM 7-6
MWXSc_Get_DoubleSCM 7-7
MWXSc_Get_IntegerSCM 7-8
MWXSc_Get_Symbol 7-9
MWXSc_Is_BooleanSCM 7-10
MWXSc_Is_DBIdSCM 7-11
MWXSc_Is_DoubleSCM 7-10
MWXSc_Is_IntegerSCM 7-11
MWXSc_Is_ListSCM 7-12
MWXSc_Is_NullSCM 7-13
MWXSc_Is_StringSCM 7-13
MWXSc_Make_BooleanSCM 7-14
MWXSc_Make_DBIdSCM 7-14
MWXSc_Make_DoubleSCM 7-15
MWXSc_Make_IntegerSCM 7-15

Chapter IN:

MWXSc_Make_ListSCM 7-16
MWXSc_Make_StringSCM 7-17
MWXSc_Register_Function 7-17
MWXSc_Scanf 7-18
MXSc_Get_StringSCM 7-8
myOpenCel function 1-29
myOpenCell function 1-29

N

naming libraries 2-2
Net object 3-50
netlist cell 2-5
notch fill cell 2-5

O

object data fields, getting 3-4
object ID 2-3
omwx_proto.h 1-12
opening a cell 2-7

P

Path object 3-54
Pin object 3-55
PinAccessEdge object 3-59
PinInstMaster object 3-60
PlanGroup object 3-62
Polygon object 3-64
Port object 3-65
PortCap Object 3-67
PortFunc object 3-67
PortInst object 3-68
PortInstMaster object 3-69
PortToPortDelay object 3-70
PortToPortTable object 3-71
PreRouteSegment object 3-50

Q

Query Hierarchical Object Example 1-20

R

Rectangle object 3-73

RectArray object 3-74

recType field 3-108

REdge object 3-74

reference library 2-2

RegionBlockage object 3-75

RLCCoupleNet object 3-75

RLCNet object 3-77

RLCNode object 3-79

RLCPort object 3-79

RouteGuide object 3-80

routeType field 3-108

S

sample C subroutines 1-28

Scheme loadModule command 1-32

statusFlag, common access field 3-109

Sun and HP dynamic linkers 1-32

symbols, defining 1-32

T

tableType field 3-110

Text object 3-97

TilePattern object 3-98

timing cell 2-5

transformCode field 3-111

U

using a new C function 1-30

Chapter IN:

IN-18

V

VerticalWire object 3-99

ViaRegion object 3-100

W

WireDirection object 3-102

WireMaster object 3-102

WireTrack object 3-104

