



Pyxis Project Manager Reference Manual

for the Pyxis Custom Design Platform

Software Version 10.5_1

© 1990-2015 Mentor Graphics Corporation
All rights reserved.

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

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

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

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

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

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

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

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

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

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

Table of Contents

Chapter 1

Function Dictionary Overview.....	17
Document Conventions	17
Available Functions	20
Function Summary.....	20
Interactive Functions.....	21
Design Object Toolkit Functions	28
Configuration Management Toolkit Functions.....	33
Tool Viewpoint Functions	38
iDM Interactive Functions	39
iDM Toolkit Functions	40
Language Flow Functions.....	41

Chapter 2

Function Dictionary Part 1	43
\$\$add_configuration_entry()	44
\$add_configuration_entry()	46
\$\$add_container()	48
\$add_container()	49
\$\$add_directory()	51
\$add_directory()	52
\$add_object_property().....	53
\$add_project_to_rc().....	55
\$\$add_reference()	57
\$add_reference()	60
\$add_reference_property()	62
\$add_toolbox()	64
\$\$add_type()	66
\$add_versions().....	68
\$browse_for_object()	70
\$\$build_configuration()	72
\$build_configuration()	74
\$cancel_checkout_objects_for_rc()	77
\$\$change_configuration_references().....	78
\$change_configuration_references().....	80
\$\$change_design_object_references()	83
\$change_design_object_references()	86
\$change_location_map_entry().....	89
\$\$change_object_name()	91
\$change_object_name()	93
\$change_object_property().....	95
\$\$change_object_references()	97

\$change_object_references()	100
\$change_password()	103
\$change_protection()	104
\$change_reference_property()	106
\$change_reference_state()	108
\$change_version_depth()	110
\$check_language_views()	112
\$check_references()	113
\$check_registries()	115
\$checkin_objects_to_rc()	117
\$checkout_objects_from_rc()	119
\$\$clear_entry_filter()	121
\$\$clear_global_status()	123
\$\$clear_monitor()	124
\$\$close_configuration()	125
\$close_hierarchy()	126
\$\$close_versioned_object()	127
\$close_window()	129
\$compile_all_models()	130
\$\$convert_configuration_references()	131
\$convert_configuration_references()	133
\$\$convert_object_references()	135
\$convert_object_references()	137
\$\$copy_configuration()	139
\$copy_configuration()	141
\$\$copy_design_object()	144
\$copy_design_object()	147
\$\$copy_object()	151
\$copy_object()	155
\$copy_version()	159
\$\$create_configuration()	162
\$create_dm_category()	163
\$create_dm_cell()	164
\$create_dm_ext_lib()	165
\$create_dm_library()	166
\$create_dm_project()	167
\$create_dm_tech_category()	168
\$create_dm_tech_lib()	169
\$create_symbols_for_hdl_lib()	170
\$create_symbols_for_src()	171
\$create_tech_config_object()	172
\$\$create_versioned_object()	174
\$create_work_area_from_rc()	176

Chapter 3

Function Dictionary Part 2 177

\$\$delete_configuration()	178
\$delete_configuration()	180

Table of Contents

\$delete_design_object()	182
\$delete_excess_versions()	184
\$\$delete_object()	186
\$delete_object()	188
\$\$delete_object_property()	190
\$delete_object_property()	192
\$\$delete_reference()	194
\$delete_reference()	196
\$\$delete_reference_handle()	198
\$\$delete_reference_property()	200
\$delete_reference_property()	202
\$\$delete_reference_property_handle()	204
\$\$delete_version()	206
\$delete_version()	207
\$\$delete_version_property()	209
\$descend_hierarchy_one_level()	210
\$descend_hierarchy_specify_level()	212
\$\$duplicate_object()	215
\$edit_file()	218
\$empty_trash()	220
\$explore_contents()	221
\$explore_parent()	222
\$explore_reference_parent()	223
\$explore_references()	225
\$export_configuration_entries()	227
\$export_library()	228
\$export_location_map()	229
\$find_external_deps()	230
\$find_references()	231
\$\$fix_relative_path()	232
\$\$freeze_configuration()	233
\$freeze_configuration()	234
\$\$freeze_version()	236
\$freeze_version()	238
\$get_area_selected_objects()	240
\$\$get_children()	242
\$\$get_configuration_entries()	244
\$\$get_configuration_path()	245
\$\$get_container_contents()	246
\$\$get_date_last_modified()	247
\$get_default_tool()	248
\$\$get_entry_version()	249
\$\$get_fileset_members()	250
\$\$get_hard_name()	251
\$\$get_location_map()	253
\$\$get_monitor_error_count()	255
\$\$get_monitor_flag()	256
\$\$get_monitor_verbosity()	258
\$\$get_monitor_warning_count()	259

\$get_navigator_directory()	260
\$get_navigator_directory_hard()	261
\$get_next_tool_env()	262
\$\$get_object_current_version()	263
\$\$get_object_parent_path()	264
\$\$get_object_path_filter()	265
\$get_object_pathname()	267
\$\$get_object_properties()	268
\$\$get_object_property_filter()	270
\$\$get_object_property_value()	272
\$\$get_object_protection()	273
\$\$get_object_references()	275
\$\$get_object_type()	277
\$get_object_type()	278
\$\$get_object_type_filter()	279
\$get_object_version()	281
\$\$get_object_versions()	282
\$\$get_parent_entry()	283
\$\$get primaries()	285
\$\$get_reference_properties()	286
\$\$get_reference_properties_handle()	288
\$\$get_reference_property_filter()	290
\$\$get_reference_traversal()	292
\$\$get_secondaries()	293
\$\$get_soft_name()	295
\$\$get_status_code()	297
\$\$get_status_code_stack()	298
\$\$get_status_messages()	299
\$get_subinvoke_mode()	300
\$\$get_target_path()	301
\$get_technology()	302
\$get_toolbox_search_path()	303
\$get_tool_pathname()	304
\$get_tool_script()	305
\$get_tool_type()	307
\$\$get_type_properties()	308
\$\$get_type_property_value()	309
\$\$get_version_depth()	310
\$\$get_version_properties()	311
\$\$get_working_directory()	313
\$goto_directory()	314
\$\$handle_map_error()	316
\$\$has_object_property()	318
\$\$has_reference_property()	320
\$\$has_reference_property_handle()	322
\$hide_monitor()	324
\$hide_secondary_entries()	325

Chapter 4

Function Dictionary Part 3 326

\$import_classic_data()	327
\$import_custom_view()	329
\$import_design_kit()	330
\$import_ext_lib()	332
\$import_hdl()	333
\$import_icstudio_library()	337
\$import_icstudio_project()	339
\$import_spice()	341
\$include_external_library()	344
\$invoke_bgd_tool()	345
\$invoke_tool()	346
\$\$is_build_consistent()	348
\$\$is_build_valid()	349
\$\$is_configuration_edited()	350
\$\$is_configuration_frozen()	351
\$\$is_configuration_locked()	352
\$\$is_container()	353
\$\$is_directory()	354
\$\$is_entry_container()	355
\$\$is_entry_fixed()	356
\$\$is_entry_primary()	357
\$\$is_entry_retargetable()	359
\$\$is_object_released()	361
\$\$is_object_versioned()	363
\$\$is_read_protected()	364
\$\$is_relative_path()	365
\$\$is_type_versioned()	366
\$\$is_writable()	367
\$\$is_write_protected()	368
\$list_references()	369
\$load_registry()	370
\$\$lock_configuration()	371
\$lock_configuration()	372
\$\$lock_object()	374
\$login_admin()	377
\$logged_in()	378
\$logout_admin()	379
\$maintain_hierarchy()	380
\$\$monitor_global_status()	381
\$\$move_design_object()	382
\$move_design_object()	386
\$\$move_object()	390
\$move_object()	392
\$\$object_complete()	394
\$\$object_exists()	395
\$object_status_for_rc()	396

\$\$open_configuration()	397
\$open_configuration_window()	399
\$\$open_hierarchy()	401
\$open_navigator()	402
\$open_object()	404
\$open_read_only_editor()	406
\$open_session_monitor()	407
\$\$open_tool()	408
\$open_tool()	410
\$open_tools_window()	411
\$open_trash_window()	412
\$open_types_window()	413
\$\$open_versioned_object()	414
\$\$prune_design_hierarchy()	416
\$\$read_map()	418
\$read_map()	420
\$refresh_all()	422
\$\$release_configuration()	423
\$release_configuration()	425
\$\$release_object()	428
\$release_object()	432
\$\$remove_configuration_entry()	435
\$remove_configuration_entry()	436
\$remove_external_library()	437
\$remove_toolbox()	438
\$report_configuration_info()	439
\$\$report_configuration_references()	441
\$report_configuration_references()	442
\$report_entry_info()	444
\$\$report_entry_verification()	446
\$report_entry_verification()	447
\$\$report_global_status()	449
\$report_object_info()	450
\$report_reference_info()	453
\$report_tool_info()	456
\$report_type_info()	458
\$report_version_info()	459
\$\$resolve_path()	462
\$revert_object_from_rc()	464
\$\$revert_version()	466
\$revert_version()	467

Chapter 5

Function Dictionary Part 4 469

\$\$salvage_object()	470
\$salvage_object()	471
\$\$save_configuration()	473
\$save_configuration()	474

Table of Contents

\$\$save_configuration_as()	475
\$save_configuration_as()	476
\$\$save_object()	477
\$save_toolbox_search_path()	479
\$search()	481
\$search_again()	483
\$select_all()	484
\$select_by_name()	485
\$select_by_library()	487
\$select_by_type()	488
\$select_config_entry()	490
\$select_object()	492
\$select_reference()	494
\$select_tool()	496
\$select_toolbox()	498
\$select_trash_object()	500
\$select_version()	502
\$set_build_rules()	504
\$set_file_compilation_options()	510
\$set_ini_mappings()	512
\$\$set_location_map_entry()	513
\$\$set_monitor_flag()	515
\$\$set_monitor_verbosity()	517
\$set_next_tool_env()	518
\$\$set_object_path_filter()	519
\$\$set_object_property()	521
\$\$set_object_property_filter()	523
\$\$set_object_type_filter()	525
\$set_project_options()	527
\$set_project_refresh_heartbeat()	529
\$set_project_registration_options()	530
\$\$set_protection()	532
\$\$set_protection_numeric()	534
\$\$set_reference_property()	536
\$\$set_reference_property_filter()	539
\$\$set_reference_property_handle()	541
\$\$set_reference_traversal()	543
\$set_subinvoke_mode()	545
\$\$set_target_path()	546
\$set_target_path()	548
\$set_technology()	550
\$set_toolbox_search_path()	551
\$\$set_version_depth()	553
\$\$set_version_property()	555
\$\$set_working_directory()	557
\$set_working_directory()	558
\$setup_filter_active()	560
\$setup_filter_all()	563
\$setup_default_editor()	566

\$setup_iconic_window_layout()	568
\$setup_invoke_tool()	570
\$\$setup_monitor()	571
\$setup_monitor()	574
\$setup_session_defaults()	577
\$setup_startup_windows()	579
\$show_all_files()	581
\$show_compiled_libs()	582
\$show_component_hierarchy()	583
\$show_custom_views()	588
\$show_directories()	589
\$show_ext_libs()	590
\$show_invalid_language_views()	591
\$show_language_views()	592
\$show_layout_views()	593
\$\$show_location_map()	594
\$show_location_map()	595
\$show_logic_views()	596
\$show_references()	597
\$show_monitor()	599
\$show_tech_libs()	600
\$show_versions()	601
\$trash_object()	602
\$\$unfreeze_configuration()	603
\$unfreeze_configuration()	604
\$\$unfreeze_version()	605
\$unfreeze_version()	607
\$\$unlock_configuration()	608
\$unlock_configuration()	609
\$\$unlock_object()	610
\$unselect_all()	612
\$unselect_by_name()	614
\$unselect_by_type()	616
\$unselect_config_entry()	618
\$unselect_object()	620
\$unselect_reference()	622
\$unselect_tool()	624
\$unselect_toolbox()	625
\$unselect_trash_object()	626
\$unselect_version()	627
\$unset_next_tool_env()	629
\$untrash_object()	630
\$update_objects_from_rc()	631
\$\$update_type()	633
\$update_window()	635
\$validate_technology()	636
\$view_by_icon()	637
\$view_by_name()	638
\$view_containment_hierarchy()	639

Table of Contents

\$view_primary_hierarchy()	640
\$view_secondary_entries()	641
\$view_toolboxes()	642
\$view_tools()	643
\$write_default_startup_file()	644
\$\$writeln_monitor()	646

Chapter 6

Integrated Pyxis Project Manager Function Dictionary..... 649

Hierarchy Window.....	649
\$get_current_obj_hier_path()	651
\$get_current_obj_inst_list()	652
\$idw_dh_setup_display()	653
\$idw_report_hier()	655
\$idw_open_hierarchy_window()	656
\$inst_area_extend_selection()	658
\$inst_area_select_all_items()	659
\$inst_area_select_item()	660
\$inst_area_show_instances()	661
\$inst_area_unselect_all_items()	662
\$make_obj_current()	663
\$open_new_comp_hierarchy()	664
\$open_new_hierarchy()	665
\$select_obj()	666
\$show_instance()	667
\$show_n_levels()	668
\$set_font()	669
\$setup_comp_hierarchy_display()	670
\$setup_hierarchy_selection()	671
Component Window.....	672
\$add_components()	674
\$add_labels_to_models()	675
\$collapse_object()	677
\$delete_labels_from_models()	678
\$delete_part_interfaces()	679
\$expand_object()	680
\$forget_components_edits()	682
\$hide_body_props()	683
\$hide_labels()	684
\$hide_model()	685
\$hide_pin_properties()	686
\$hide_pins()	687
\$register_models()	688
\$remove_components()	690
\$rename_part_interface()	691
\$report_body_prop_info()	692
\$report_component_info()	693
\$report_model_entry_info()	694

\$report_models_for_each_label()	695
\$report_model_info()	696
\$report_models_with_all_labels()	697
\$report_pin_info()	698
\$save_components_edits()	699
\$select_model_object()	700
\$select_object()	702
\$set_bgd_color()	704
\$set_bgd_color_title_items()	705
\$set_bgd_color_titles()	706
\$set_constraints()	707
\$set_default_part_interface()	708
\$set_fgd_color()	709
\$set_fgd_color_title_items()	710
\$set_fgd_color_titles()	711
\$set_font()	712
\$set_part_interface_font()	713
\$show_body_props()	714
\$show_labels()	715
\$show_model()	716
\$show_pin_properties()	717
\$show_pins()	718
\$unselect_model_object()	719
\$unselect_object()	720
\$validate_models()	722

Chapter 7

Shell Command Dictionary 723

Common Shell Command Usage	724
Finding the Name and Type of a Design Object in the Shell	724
Switches, Labels, and Names	725
About Supplying Arguments on Standard Input	726
About Supplying Options to Design Management Shell Commands	727
Object Specifiers	728
Error Handling	729
Shell Command Summary	730
change_ncf_ref	732
change_references	734
checkref	737
chref	738
copy_object	740
copy_version	742
ddms_locenv	745
ddms_which_map	747
delete_object	749
dmgr_ic	751
freeze_version	753
get_hard_name	755

Table of Contents

get_soft_name	757
list_contents	759
listref	762
list_references	763
move_object	766
revert_version	769
salvage_object	771
set_version_depth	773
show_object_info	775
unfreeze_version	777
 Appendix A	
AMPLE Data Types	779
Common Data Types	779
 Appendix B	
Toolkit Examples	781
Design Object Toolkit Examples	781
Configuration Management Toolkit Examples	783
Qualification Script Example	787
 Appendix C	
Regular Expressions	791
UNIX System V Regular Expressions	791
Wildcards	793
 Appendix D	
Function Cross-Reference	795
Commands to Functions	795
Menu Paths to Functions	796
Logical Key Names Mapped to Functions and Scopes	803
Logical Key Names Mapped to Physical Keys	806
 Third-Party Information	
End-User License Agreement	

List of Figures

Figure 5-1. Set Build Rules Dialog Box 508

List of Tables

Table 1-1. Conventions for Command Usage Syntax	18
Table 1-2. Summary of the Interactive Functions	21
Table 1-3. Summary of Design Object Toolkit Functions	29
Table 1-4. Summary of the Configuration Management Toolkit Functions	35
Table 1-5. Summary of the Tool Viewpoint Functions	39
Table 1-6. Summary of iDM Interactive Functions	40
Table 1-7. Summary of iDM Toolkit Functions	41
Table 1-8. Language Flow Functions	41
Table 6-1. Summary of Hierarchy Window Functions	649
Table 6-2. iDM Component Window Functions	672
Table 7-1. Shell Commands	730
Table A-1. Common Data Types	779
Table C-1. UNIX System V Regular Expressions	791
Table C-2. Pyxis Project Manager Functions Accepting Regular Expressions	792
Table C-3. Wildcards	793
Table C-4. Pyxis Project Manager Functions Accepting Wildcards	793
Table D-1. Pulldown Menu Paths Mapped to Functions	796
Table D-2. Popup Menu Paths Mapped to Functions	800
Table D-3. Key Names Mapped to Functions & Scopes	804
Table D-4. Logical Key Names for HP Keyboards	806
Table D-5. Logical Key Names for Sun Keyboards	810

Chapter 1

Function Dictionary Overview

This chapter lists the functions available in the Pyxis Project Manager application. It also lists most of the functions that are available for Integrated Design Management tasks.

You can find Integrated Design Management functions for the Hierarchy window and the Component window in [“Integrated Pyxis Project Manager Function Dictionary”](#) on page 649.

The following topics describe the conventions for this document and a list of available Pyxis Project Manager functions.

Document Conventions	17
Available Functions	20
Function Summary	20

Document Conventions

The conventions used to present information in the Pyxis documentation are described in the following sections.

- [“Function Syntax”](#) on page 17.
- [“Command Usage Syntax”](#) on page 18.
- [“Menu Paths”](#) on page 19.
- [“Linux Pathnames”](#) on page 19.
- [“Transcribed Examples”](#) on page 19.
- [“Notational Information”](#) on page 19.

Function Syntax

The AMPLE application functions are structured, and this section describes the AMPLE syntax.

Example 1-1. AMPLE Application Function

```
$function_name(arg_1, arg_2, arg_3, arg_4)
```

Note the following in [Example 1-1](#):

- The name of the function always begins with a dollar sign or, sometimes for built-in functions, two dollar signs. In this example, “\$function_name” is the name of the function.
- Regular font indicates a required argument. In this example, `arg_1` and `arg_2` are required arguments.
- Italic font indicates an optional argument. In this example, *arg_3* and *arg_4* are optional arguments.

For more information on function syntax, refer to the “[Function Call Syntax and Semantics](#)” section in the *AMPLE for Pyxis User's Manual*.

Command Usage Syntax

The shell command and function usage syntax for AMPLE commands follows pre-set conventions. AMPLE commands are found in the Pyxis reference manuals.

[Table 1-1](#) describes the usage conventions for the command syntax in the Usage section of the command documentation.

Table 1-1. Conventions for Command Usage Syntax

Convention	Example	Usage
UPPercase	REPort ENvironment (substitute product-specific examples throughout)	Required command letters are in uppercase; in most cases, you may omit lowercase letters when entering commands or literal arguments and you need not enter in uppercase. Command names and options are normally case insensitive, but for some tools the invocation command name is case sensitive and must be lowercase. Commands usually follow the 3-2-1 rule: the first three letters of the first word, the first two letters of the second word, and the first letter of the third, fourth, etc. words.
<i>Italic</i>	<i>“comment”</i>	Italics indicate that the argument is optional. Required arguments are shown in regular text in the usage syntax.
“”	“target”	Quotation marks indicate a string value for the argument.
[]	[[“object1”, “type1”], [“object2”, “type2”]]	Square brackets indicate a vector or location value for the argument.

Menu Paths

The following conventions are used to specify a location in the graphical user interface:

Menu1 > Menu2 > Item

The greater-than symbol (>) separates the menu name from either the next level of the menu hierarchy (Menu2) or from a menu selection (Item).

Sometimes, the name of the window in which the menu appears is shown first:

(Window) Menu1 > Menu2 > Item

When you can select one of several menu items to perform a similar function, a vertical bar separates the choices:

Menu1 > Menu2 > Alternative1 | Alternative2 | Alternative3

Linux Pathnames

Most pathnames are prepended by soft prefixes, such as *\$MGC_GENLIB* and *\$PROJECT_XYZ*, which indicate shared network resources. The dollar sign signifies the beginning of the soft prefix and the rest of the prefix is presented in capital letters. In the following example, *\$MGC_HOME* is the soft prefix:

\$MGC_HOME/pkgs/dme/userware/default

In instructional text, pathnames are shown in bold monospace typeface. The generic part for which you are to make a substitution is shown in standard type. In the following example, you would substitute a specific package name for the generic phrase “pkg_name” when typing the pathname at the command line:

\$MGC_HOME/pkgs/pkg_name/userware/default

Transcribed Examples

Transcribed examples are presented in monospaced font. Within transcribed examples, bold text indicates what your response should be to an included instruction:

```
Enter target directory for ic_parts [For example  
/net/machine_1/home]> path_to_ic_parts_dir
```

```
MAP library memory_lib_map will be installed in  
/path_to_ic_parts_dir.
```

Notational Information

Three kinds of notational information is emphasized by setting it apart from the standard text:



Caution

A **Caution** marks situations that are unwanted or undefined such as recommending you not store a permanent file in */tmp*. They also alert you to situations that could cause serious or dangerous consequences, such as an action that deletes or overwrites a file.

Note

A **Note** flags important information that may be of special interest such as a prompt reminding you to make backup copies of files



Tip: directs you to additional information on the topic, typically in a different chapter of this manual or in a different manual.

Available Functions

While using Pyxis Project Manager application, you have access to user interface, printer, and AMPLE functions and commands. These additional functions let you modify the user interface to meet your needs. You can use them to create and manipulate windows, set up a default printing environment, automate functionality, or change colors and fonts.

For complete descriptions of user interface, printer, and AMPLE functions and commands, refer to the following related manuals:

- *Common User Interface Reference Manual* — This manual contains detailed reference information about all of the Common User Interface functions.
- *AMPLE for Pyxis Reference Manual* — This manual contains detailed reference information about AMPLE statements and functions common to all applications.
- *AMPLE for Pyxis User's Manual* — This programming manual provides information, examples, and procedures for writing AMPLE scripts. Topics include function declaration, data types, scope, input and output, flow of control, transcripts, and startup files.
- *Customizing the Pyxis Common User Interface* — This manual tells how to extend the user interface, and is available on SupportNet only. It explains how to redefine keys and how to create your own menus, windows, dialog boxes, messages, and palettes.

Function Summary

Each table contains an alphabetized listing of functions and a short description of each function.

Interactive Functions.	21
Design Object Toolkit Functions	28
Configuration Management Toolkit Functions.	33
Tool Viewpoint Functions	38
iDM Interactive Functions	39
iDM Toolkit Functions	40
Language Flow Functions.	41

Interactive Functions

Interactive functions are functions that map directly to the Pyxis Project Manager menu selections and commands. Most of these functions assume that you have selected at least one design object on which to operate.

Functions that return pathnames always return soft pathnames, if possible. You can determine what hard pathname the soft pathname maps to, by using the `$$get_hard_name()` function. For information on hard, soft, relative, and absolute pathnames, refer to the *Pyxis Project Manager User's Manual*.

Not all of the interactive functions can operate in all of the Pyxis Project Manager windows. Each function's reference page lists the function's scope and, under "Prerequisites," the window that must be active for you to execute the function.

Note



The `$set_active_window()` enables you to specify which window is active. If you execute a macro from the popup command line that calls the `$close_window()` function, the macro's next line must call the `$set_active_window()` function; otherwise, no window is active and the macro fails.

Table 1-2 describes all of the Pyxis Project Manager interactive functions.

Table 1-2. Summary of the Interactive Functions

Function	Description
<code>\$add_configuration_entry()</code>	Adds the current version of a design object to the active Configuration window.
<code>\$add_container()</code>	Creates a basic container design object with an attribute file.
<code>\$add_directory()</code>	Creates a directory without an attribute file.
<code>\$add_object_property()</code>	Adds a property to the selected design object.
<code>\$add_reference()</code>	Creates a reference from the selected design object to a target design object.
<code>\$add_reference_property()</code>	Adds a property to the selected reference.
<code>\$add_toolbox()</code>	Appends the specified toolbox to the current toolbox search path.
<code>\$add_versions()</code>	Adds the specified versions of the selected design object to the active Configuration window.
<code>\$browse_for_object()</code>	Brings up a dialog navigator.

Table 1-2. Summary of the Interactive Functions (cont.)

Function	Description
\$build_configuration()	Adds secondary entries to the active Configuration window, based on the build rules of the primary entries.
\$change_configuration_references()	Changes the references of each entry in the active Configuration window.
\$change_location_map_entry()	Overrides a location map entry in memory.
\$change_object_name()	Renames the specified design object.
\$change_object_property()	Changes a property value of the selected design object.
\$change_object_references()	Changes the references of the selected design object.
\$change_protection()	Changes the protection of the selected design object.
\$change_reference_property()	Changes a property of the selected reference.
\$change_reference_state()	Changes the state of the selected reference.
\$change_version_depth()	Changes the version depth of the selected design object.
\$check_references()	Provides a simplified interface to check and fix the broken references of the selected design objects.
\$check_registries()	Checks the \$MGC_HOME/registry directory.
\$close_window()	Closes the active window.
\$convert_configuration_references()	Automatically converts reference pathnames on all objects in the configuration from hard reference pathnames to soft reference pathnames using the active location map.
\$convert_object_references()	Automatically converts the reference pathnames of selected design objects from hard to soft using the current location map.
\$copy_configuration()	Copies all design object versions in the active Configuration window to a target location.
\$copy_object()	Copies the selected design objects to a target location.
\$copy_version()	Copies a single selected version of the design object to a target location.

Table 1-2. Summary of the Interactive Functions (cont.)

Function	Description
\$delete_configuration()	Deletes all design object versions in the active Configuration window from the disk.
\$delete_excess_versions()	Deletes all but a specified number of versions of the selected design object(s).
\$delete_object()	Deletes the selected design object.
\$delete_object_property()	Deletes the specified property from the selected design object.
\$delete_reference()	Deletes the selected reference from the source design object.
\$delete_reference_property()	Deletes the specified property from the selected reference.
\$delete_version()	Deletes the selected design object version.
\$edit_file()	Opens a file in edit or read-only mode.
\$empty_trash()	Deletes the contents of the Trash window.
\$explore_contents()	Navigates down one level of the containment hierarchy.
\$explore_parent()	Navigates up one level of the containment or reference network.
\$explore_reference_parent()	Navigates to and displays the contents of the parent directory of the referenced object.
\$explore_references()	Navigates down one level of the reference network.
\$find_references()	Finds the references for the selected part.
\$freeze_configuration()	Freezes all design object versions in the active Configuration window and creates a new version of the configuration.
\$freeze_version()	Freezes the selected design object version.
\$get_area_selected_objects()	Returns the pathnames and types of the selected design objects in the Navigator window.
\$get_default_tool()	Returns the default tool for the specified type.
\$get_navigator_directory()	Returns the soft pathname of the current navigator directory.
\$get_navigator_directory_hard()	Returns the hard pathname of the current navigator directory.
\$get_toolbox_search_path()	Returns the current toolbox search path.

Table 1-2. Summary of the Interactive Functions (cont.)

Function	Description
\$goto_directory()	Navigates to the specified directory.
\$hide_secondary_entries()	Hides the secondary entries in the active Configuration window.
\$hide_monitor()	Hides the configuration's monitor window.
\$list_references()	List the all references for a component.
\$load_registry()	Loads a type registry into the current session's type manager.
\$lock_configuration()	Locks all design object versions in the active Configuration window.
\$maintain_hierarchy()	Maintains a parts directory hierarchy upon release.
\$move_object()	Moves the selected design object to a target location.
\$open_configuration_window()	Opens a new or existing configuration object.
\$open_navigator()	Opens a navigator.
\$open_object()	Opens the selected design object.
\$open_read_only_editor()	Invokes the read-only ASCII editor on the selected object.
\$open_session_monitor()	Makes the session monitor window visible.
\$open_tool()	Invokes the selected tool.
\$open_tools_window()	Opens a Tools window.
\$open_trash_window()	Opens a Trash window.
\$open_types_window()	Displays the types currently loaded in the Pyxis Project Manager application.
\$read_map()	Reads the ASCII location map file into memory.
\$release_configuration()	Releases the design object versions in the active Configuration window to a target location.
\$release_object()	Releases the current version of the selected objects to the specified destination directory.
\$remove_configuration_entry()	Removes an entry from the active Configuration window.
\$remove_toolbox()	Removes a toolbox from the toolbox search path.

Table 1-2. Summary of the Interactive Functions (cont.)

Function	Description
\$report_configuration_info()	Displays the status of the active Configuration window.
\$report_configuration_references()	Reports the references of entries in the active Configuration window.
\$report_entry_info()	Displays the status of the selected entries in the active Configuration window.
\$report_entry_verification()	Reports the integrity of entries in the active configuration.
\$report_object_info()	Displays information about the selected design object.
\$report_reference_info()	Displays information about the selected reference.
\$report_tool_info()	Displays information about the selected tool.
\$report_version_info()	Displays information about the selected version.
\$report_type_info()	Displays the properties of the selected type.
\$revert_version()	Deletes the current version of the selected design object and makes the most recent version the new current version.
\$salvage_object()	Attempts to repair a design object that has existing session data or malformed locks resulting from an application failure.
\$save_configuration()	Writes the entries in the active Configuration window to disk and increments the version number of the configuration object.
\$save_configuration_as()	Saves the entries in the active configuration under a new configuration object name.
\$save_toolbox_search_path()	Saves the toolbox search path.
\$search()	Searches the active window for the specified pattern.
\$search_again()	Repeats the search of the most recent search pattern specified.
\$select_all()	Selects all design objects in the active window.
\$select_by_name()	Selects a design object based on its name.
\$select_by_library()	Selects specific parts libraries in a configuration.
\$select_by_type()	Selects a design object based on its type.

Table 1-2. Summary of the Interactive Functions (cont.)

Function	Description
\$select_config_entry()	Selects a configuration entry.
\$select_object()	Selects a design object.
\$select_reference()	Selects a reference.
\$select_tool()	Selects a tool.
\$select_toolbox()	Selects a toolbox.
\$select_trash_object()	Selects a trash object.
\$select_version()	Selects a design object version.
\$set_build_rules()	Sets the build rules for the selected primary entries.
\$set_target_path()	Sets the target path of selected configuration entries in the active Configuration window.
\$set_toolbox_search_path()	Sets the order of the toolbox search path.
\$set_working_directory()	Changes the value of the MGC working directory.
\$setup_filter_active()	Specifies filters that determine which objects are displayed in the active navigator.
\$setup_filter_all()	Specifies filters that determine which objects are displayed in all navigators that you open after executing this function.
\$setup_default_editor()	Sets the default editor.
\$setup_iconic_window_layout()	Specifies the layout features in iconic windows.
\$setup_monitor()	Specifies the setup of the configuration monitoring facilities.
\$setup_session_defaults()	Specifies the session defaults for the Pyxis Project Manager graphical interface.
\$setup_startup_windows()	Specifies which windows are opened at invocation.
\$show_location_map()	Displays, in a read-only window, the absolute path of the current location map, and the value of its entries.
\$show_references()	Displays the references of the selected design object.
\$show_monitor()	Makes the configuration monitor window visible.

Table 1-2. Summary of the Interactive Functions (cont.)

Function	Description
\$show_versions()	Displays the versions of the selected design object.
\$trash_object()	Moves the selected design object to the Trash window.
\$unfreeze_configuration()	Unfreezes all design object versions in the active Configuration window.
\$unfreeze_version()	Unfreezes a frozen version of the selected design object.
\$unlock_configuration()	Unlocks all design object versions in the active Configuration window.
\$unselect_all()	Unselects all selected design objects.
\$unselect_by_name()	Unselects a design object based on its name.
\$unselect_by_type()	Unselects a design object based on its type.
\$unselect_config_entry()	Unselects all selected configuration entries.
\$unselect_object()	Unselects the selected design object.
\$unselect_reference()	Unselects the selected reference.
\$unselect_tool()	Unselects the selected tool.
\$unselect_toolbox()	Unselects the selected toolbox.
\$unselect_trash_object()	Unselects the selected trash objects.
\$unselect_version()	Unselects the selected version.
\$untrash_object()	Removes the selected design object from the Trash window.
\$update_window()	Updates the currently active window with the most current information.
\$view_by_icon()	Places the current navigator in iconic-viewing mode.
\$view_by_name()	Places the current navigator in list-viewing mode.
\$view_containment_hierarchy()	Displays the containment hierarchy of all entries in the active Configuration window.
\$view_primary_hierarchy()	Displays the primary entries in the active Configuration window.
\$view_secondary_entries()	Displays the secondary entries in the active Configuration window.

Table 1-2. Summary of the Interactive Functions (cont.)

Function	Description
\$view_toolboxes()	Changes the viewing mode of the Tools window to display the toolboxes in the toolbox search path.
\$view_tools()	Changes the viewing mode of the Tools window to display the current set of tool icons.
\$write_default_startup_file()	Saves the values of the current session settings to the default startup file.

Design Object Toolkit Functions

A toolkit is a set of functions that you use to write scripts to supplement the capabilities of the Pyxis Project Manager application and to perform operations in batch mode. The functions in the design object toolkit enable you to write scripts that manipulate design objects.

To see example scripts that use design object toolkit functions, refer to “[Design Object Toolkit Examples](#)” on page 781.

For information about Pyxis Project Manager functions you can invoke using menus and the command line, refer to “[Interactive Functions](#)” on page 21.

Toolkit Functions for Versioning Third-Party Data

The design object toolkit contains three special purpose functions which provide a mechanism for versioning design objects produced with non-Mentor Graphics applications — `$$create_versioned_object()`, `$$open_versioned_object()`, and `$$close_versioned_object()`. These functions enable you to version encapsulated data in conformance with the Mentor Graphics versioning scheme, without having to recompile the non-Mentor Graphics tool(s).

For a complete description of each of these functions, refer to the function's corresponding reference page in this manual.

For example qualification and termination scripts that use these three special purpose functions, and for complete information about encapsulating tools and design data, refer to “[Example Encapsulation](#)” in the *Pyxis Registry User's and Reference Manual*.

Specifying a Design Object's Pathname

When you specify a design object's pathname, the design object toolkit functions enable you to specify either the absolute or the relative pathname of the object. In a Navigator window, a relative pathname is relative to the current navigator directory.

You can query the value of the current navigator directory by using the [\\$get_navigator_directory\(\)](#) and [\\$get_navigator_directory_hard\(\)](#) functions. In all other Pyxis

Project Manager windows, a relative pathname is relative to the *MGC working directory* which is set to the value of the MGC_WD environment variable. You can query or change the value of your working directory, by using the [\\$\\$get_working_directory\(\)](#) and the [\\$\\$set_working_directory\(\)](#) functions.

Functions that return pathnames always return soft pathnames, if possible. You can determine the hard pathname a soft pathname maps to, by using the [\\$\\$get_hard_name\(\)](#) function. You can determine the soft pathname a hard pathname maps to, by using the [\\$\\$get_soft_name\(\)](#) function. For information on hard, soft, relative, and absolute pathnames, refer to “[Working with References](#)” and “[Glossary](#)” in the *Pyxis Project Manager User's Manual*.

Table 1-3 describes all design object toolkit functions.

Table 1-3. Summary of Design Object Toolkit Functions

Function	Description
\$\$add_directory()	Creates a container design object without an attribute file.
\$\$add_reference()	Creates a reference from the specified source design object to the specified target design object.
\$\$clear_global_status()	Clears the global toolkit status stack.
\$\$close_versioned_object()	Closes a versioned design object that encapsulates data from a non-Mentor Graphics application.
\$\$create_versioned_object()	Creates a versioned design object that encapsulates data from a non-Mentor Graphics application.
\$\$delete_object()	Deletes the specified design object.
\$\$delete_object_property()	Deletes the specified property from the specified design object.
\$\$delete_reference()	Deletes the specified reference from the specified source design object.
\$\$delete_reference_handle()	Deletes a reference from the specified design object by using the reference target handle.
\$\$delete_reference_property()	Deletes a property from the specified reference.
\$\$delete_reference_property_handle()	Deletes a property from the specified reference by using the reference target handle.
\$\$delete_version()	Deletes the specified version of the specified design object.

Table 1-3. Summary of Design Object Toolkit Functions (cont.)

Function	Description
<code>\$\$delete_version_property()</code>	Deletes a property from the specified version.
<code>\$\$fix_relative_path()</code>	Generates an absolute pathname from a relative pathname.
<code>\$\$freeze_version()</code>	Freezes a version of the specified design object.
<code>\$\$get_container_contents()</code>	Returns the name, type, and version of the objects in the specified container.
<code>\$\$get_date_last_modified()</code>	Returns the date that the specified design object was last saved to disk.
<code>\$\$get_fileset_members()</code>	Returns the fileset members of the specified design object.
<code>\$\$get_hard_name()</code>	Returns the hard pathname equivalent of any pathname.
<code>\$\$get_location_map()</code>	Returns the absolute path of the current location map, and the values of its entries.
<code>\$\$get_object_current_version()</code>	Returns the current version of the specified design object.
<code>\$\$get_object_parent_path()</code>	Returns the pathname of the parent of the specified design object.
<code>\$\$get_object_properties()</code>	Returns the properties of the specified design object.
<code>\$\$get_object_property_filter()</code>	Returns a property value of the specified design object.
<code>\$\$get_object_protection()</code>	Returns the protection status of the specified design object.
<code>\$\$get_object_references()</code>	Returns the references of the specified design object.
<code>\$\$get_object_type()</code>	Returns the design object type of the specified design object.
<code>\$\$get_object_versions()</code>	Returns the versions of the specified design object.
<code>\$\$get_type_properties()</code>	Returns all the properties associated with that type.
<code>\$\$get_type_property_value()</code>	Returns the value of the property type.
<code>\$\$get_reference_properties()</code>	Returns the properties of the specified reference.

Table 1-3. Summary of Design Object Toolkit Functions (cont.)

Function	Description
\$\$get_reference_properties_handle()	Returns the properties of the specified reference by using the target reference handle.
\$\$get_soft_name()	Returns the soft pathname equivalent of a pathname.
\$\$get_status_code()	Returns the error message code at the top of the status stack.
\$\$get_status_code_stack()	Returns all of the error message codes on the status code stack.
\$\$get_status_messages()	Returns the error messages on the status stack.
\$\$get_version_depth()	Returns the version depth of the specified design object.
\$\$get_version_properties()	Returns the properties of the specified design object version.
\$\$get_working_directory()	Returns the value of the MGC working directory.
\$\$handle_map_error()	Lets the user determine how to proceed when an error occurs in reading a location map.
\$\$has_object_property()	Checks whether or not a property exists on the specified design object.
\$\$has_reference_property()	Checks whether or not a property exists on the specified reference.
\$\$has_reference_property_handle()	Checks whether or not a property exists on the specified reference by using the target reference handle.
\$\$is_container()	Checks whether or not the specified design object is a container.
\$\$is_directory()	Checks whether or not the specified design object is a directory.
\$\$is_object_released()	Checks whether or not the specified design object is in a released state.
\$\$is_object_versioned()	Checks whether or not the specified design object is versioned.
\$\$is_read_protected()	Checks whether or not the specified design object can be opened for reading.
\$\$is_relative_path()	Indicates whether or not the specified pathname is a relative pathname.

Table 1-3. Summary of Design Object Toolkit Functions (cont.)

Function	Description
<code>\$\$is_type_versioned()</code>	Checks whether or not the type is versioned.
<code>\$\$is_writable()</code>	Checks whether or not the current process can write to the specified design object.
<code>\$\$is_write_protected()</code>	Checks whether or not the specified design object can be opened for writing.
<code>\$\$lock_object()</code>	Locks the specified design object.
<code>\$\$monitor_global_status()</code>	Reports any errors currently on the global toolkit status stack.
<code>\$\$object_complete()</code>	Checks whether or not the specified design object is complete.
<code>\$\$object_exists()</code>	Checks whether or not the specified design object exists.
<code>\$\$open_tool()</code>	Invokes the specified tool.
<code>\$\$open_versioned_object()</code>	Opens a versioned design object that encapsulates data from a non-Mentor Graphics application.
<code>\$\$read_map()</code>	Reads the ASCII location map file into memory.
<code>\$\$report_global_status()</code>	Reports any errors currently on the global toolkit status stack to the transcript window.
<code>\$\$resolve_path()</code>	Resolves a specified pathname into a hard pathname that does not have a symbolic link in its path.
<code>\$\$revert_version()</code>	Deletes the current version of the specified design object and makes the most recent version the new current version.
<code>\$\$salvage_object()</code>	Attempts to repair a design object that has existing session data or malformed locks resulting from an application failure.
<code>\$\$save_object()</code>	Saves the specified locked design object.
<code>\$\$set_location_map_entry()</code>	Overrides a location map entry in memory.
<code>\$\$set_object_property()</code>	Sets the property of the specified design object.
<code>\$\$set_protection()</code>	Sets the protection of the specified design object.

Table 1-3. Summary of Design Object Toolkit Functions (cont.)

Function	Description
\$\$set_protection_numeric()	Sets the protection of the specified design object by using an integer.
\$\$set_reference_property()	Sets the property of the specified reference.
\$\$set_reference_property_handle()	Sets the property of the specified reference by using the target reference handle.
\$\$set_version_depth()	Sets the version depth of specified design object.
\$\$set_version_property()	Sets the property of the specified version.
\$\$set_working_directory()	Changes the value of the MGC working directory.
\$\$show_location_map()	Displays, in a read-only window, the absolute path of the current location map, and the value of its entries.
\$\$unfreeze_version()	Unfreezes a frozen version of the specified design object.
\$\$unlock_object()	Unlocks the specified design object.

Configuration Management Toolkit Functions

A toolkit is a set of functions that you use to write scripts to supplement the capabilities of the Pyxis Project Manager application and to perform operations in batch mode.

The functions in the configuration management toolkit enable you to write scripts to build, copy, and release *configurations*. A configuration is a collection of design objects that meet the criteria of a set of rules.

The configuration toolkit functions can be used in combination with the design object toolkit functions. For example, to check for errors after building a configuration, use the design object toolkit's error handling functions. For more information about the design object toolkit, refer to [“Design Object Toolkit Functions”](#) on page 28.

By default, the configuration toolkit functions report real-time status information to the session monitor window, which is a read-only window that is viewed through the pulldown menu item **Windows > Open Session Monitor**. The information in the session monitor window includes real-time status on progress, failures, errors, and warnings. Similar to the configuration monitor window, the session monitor window clears each time you issue a new configuration operation. If you want to save the information displayed in the session monitor window, you can use the Notepad facility **Export**.

Note



Not all of the toolkit functions report status information. For example, the [\\$\\$set_target_path\(\)](#) function does not report status information. To find out whether a toolkit function reports status information, refer to "Description" on that function's reference page.

If your session setup values specify to show the session monitor window, the session monitor window is opened when you execute these configuration toolkit functions.

You can also specify that configuration toolkit functions report status monitoring information to the transcript window, instead of to the session monitor window. For information about specifying your session setup values, refer to “[Changing Your Session Setup](#)” in the *Pyxis Project Manager User's Manual*.

To see example scripts that use configuration management toolkit functions, refer to “[Configuration Management Toolkit Examples](#)” on page 783. For information about Pyxis Project Manager functions you can invoke using menus and the command line, refer to “[Interactive Functions](#)” on page 21.

For information about the key concepts of configuration management, refer to the *Pyxis Project Manager User's Manual*. For procedural information about managing configurations, refer to “[Managing Designs](#)” in the *Pyxis Project Manager User's Manual*.

About Session Configuration

A session configuration is a configuration that you can manage without a Configuration window. Because the session configuration does not use a Configuration window, you perform all management operations in batch mode using the configuration management toolkit functions. You can open only one session configuration at a time. You use the `$$open_configuration()` and `$$create_configuration()` functions to initialize the session configuration. These two functions, along with the `$$close_configuration()` function, *are not available* in the Configuration window.

The functions in the configuration toolkit operate on the configuration in the active Configuration window. If no Configuration window is active and a session configuration exists, the toolkit functions operate on the session configuration.

Design Object Pathname Specifications

When you specify a design object's pathname, most of the configuration management toolkit functions enable you to specify either the absolute or relative pathname of the object.

In a Navigator window, a relative pathname is relative to the *current navigator directory*. You can query the value of the current navigator directory by using the [\\$get_navigator_directory\(\)](#) and [\\$get_navigator_directory_hard\(\)](#) functions.

In all other Pyxis Project Manager windows, a relative pathname is relative to the *MGC working directory*, which is initially set to the directory from which you invoke the Pyxis Project Manager application. You can query or change the value of your working directory, by using the [\\$\\$get_working_directory\(\)](#) and the [\\$\\$set_working_directory\(\)](#) functions.

Functions that return pathnames always return soft pathnames, if possible. You can determine what hard pathname the soft pathname maps to, by using the [\\$\\$get_hard_name\(\)](#) function. You can determine what soft pathname the hard pathname maps to, by using the [\\$\\$get_soft_name\(\)](#) function. For information on hard, soft, relative, and absolute pathnames, refer to Working with References and “Glossary” in the *Pyxis Project Manager User's Manual*.

Table 1-4 describes all configuration management toolkit functions.

Table 1-4. Summary of the Configuration Management Toolkit Functions

Function	Description
\$\$add_configuration_entry()	Adds the specified design object version to the active configuration.
\$\$add_container()	Creates a basic container design object with an attribute file.
\$\$build_configuration()	Adds secondary entries to the active configuration, based on the build rules of the primary entries.
\$\$change_configuration_references()	Changes the references of each entry in the active configuration.
\$\$change_object_name()	Renames the specified design object.
\$\$change_object_references()	Changes the references of the specified design object.
\$\$clear_entry_filter()	Resets all filters of a primary entry's build rules.
\$\$clear_monitor()	Clears all text from the configuration's monitor window.
\$\$close_configuration()	Removes the lock on the active configuration.
\$\$convert_configuration_references()	Converts the references of each entry in the active configuration.
\$\$convert_object_references()	Converts the references of the specified design object(s).
\$\$copy_configuration()	Copies all design object versions in the active configuration, to a target location.
\$\$copy_object()	Copies the specified design object to a target location.

Table 1-4. Summary of the Configuration Management Toolkit Functions

Function	Description
<code>\$\$create_configuration()</code>	Creates a configuration object.
<code>\$\$delete_configuration()</code>	Deletes all design object versions in the active configuration from the disk.
<code>\$\$duplicate_object()</code>	Duplicates a design object.
<code>\$\$freeze_configuration()</code>	Freezes all design object versions in the active configuration.
<code>\$\$get_children()</code>	Returns the children of a configuration entry.
<code>\$\$get_configuration_entries()</code>	Returns all entries in the active configuration.
<code>\$\$get_configuration_path()</code>	Returns the soft pathname of the active configuration.
<code>\$\$get_entry_version()</code>	Returns the version number of the target object of a current entry in the active configuration.
<code>\$\$get_monitor_error_count()</code>	Returns the number of errors processed by the monitor.
<code>\$\$get_monitor_flag()</code>	Returns the value of the specified configuration monitoring attribute.
<code>\$\$get_monitor_verbosity()</code>	Returns the value of the specified configuration monitoring attribute.
<code>\$\$get_monitor_warning_count()</code>	Returns the number of warnings processed by the monitor.
<code>\$\$get_object_path_filter()</code>	Returns the value of the object path filter for the specified primary entry.
<code>\$\$get_object_property_filter()</code>	Returns the value of the object property filter for the specified primary entry.
<code>\$\$get_object_type_filter()</code>	Returns the value of the object type filter for the specified primary entry.
<code>\$\$get_parent_entry()</code>	Returns the parent of the specified configuration entry.
<code>\$\$get primaries()</code>	Returns the primary entries in the active configuration.
<code>\$\$get_reference_property_filter()</code>	Returns the value of the reference property filter for the specified primary entry.
<code>\$\$get_reference_traversal()</code>	Returns the reference traversal of the specified primary entry.

Table 1-4. Summary of the Configuration Management Toolkit Functions

Function	Description
\$\$get_secondaries()	Returns the secondary entries of the specified primary entry.
\$\$get_target_path()	Returns the target path of the specified primary configuration entry.
\$\$is_build_consistent()	Checks whether or not the active configuration is consistent.
\$\$is_build_valid()	Checks whether or not the active configuration is valid.
\$\$is_configuration_edited()	Checks whether or not the active configuration has been changed and needs to be saved.
\$\$is_configuration_frozen()	Checks whether or not the active configuration is frozen.
\$\$is_configuration_locked()	Checks whether or not the active configuration is locked.
\$\$is_entry_container()	Checks whether or not the specified configuration entry is a container.
\$\$is_entry_fixed()	Checks whether or not the specified configuration entry is a fixed entry.
\$\$is_entry_primary()	Checks whether or not the specified configuration entry is a primary entry.
\$\$is_entry_retargetable()	Checks whether or not the specified configuration entry is retargetable.
\$\$lock_configuration()	Locks all design object versions in the active configuration.
\$\$move_object()	Moves the specified design objects to a target location.
\$\$open_configuration()	Opens the specified configuration object.
\$\$prune_design_hierarchy()	Prunes off back versions of a design object.
\$\$release_configuration()	Releases all design object versions in the active configuration to a target location.
\$\$release_object()	Releases the current version of the specified objects to the specified destination directory.
\$\$remove_configuration_entry()	Removes the specified entry from the active configuration.

Table 1-4. Summary of the Configuration Management Toolkit Functions

Function	Description
<code>\$\$report_configuration_references()</code>	Reports the references of entries in the active configuration.
<code>\$\$report_entry_verification()</code>	Reports the integrity of entries in the active configuration.
<code>\$\$save_configuration()</code>	Writes the active configuration to the disk and increments the version number of the configuration object.
<code>\$\$save_configuration_as()</code>	Saves the active configuration under a new name.
<code>\$\$set_monitor_flag()</code>	Changes a single attribute of the configuration monitoring setup.
<code>\$\$set_monitor_verbosity()</code>	Sets the configuration monitor verbosity level.
<code>\$\$set_object_path_filter()</code>	Sets the value of the object path filter for the specified primary entry.
<code>\$\$set_object_property_filter()</code>	Sets the value of the object property filter for the specified primary entry.
<code>\$\$set_object_type_filter()</code>	Sets the value of the object type filter for the specified primary entry.
<code>\$\$set_reference_property_filter()</code>	Sets the value of the reference property filter for the specified primary entry.
<code>\$\$set_reference_traversal()</code>	Sets the reference traversal of the specified primary entry.
<code>\$\$set_target_path()</code>	Sets the target path of the specified configuration entries.
<code>\$\$setup_monitor()</code>	Specifies the setup of configuration monitoring facilities.
<code>\$\$unfreeze_configuration()</code>	Unfreezes all design object versions in the active configuration.
<code>\$\$unlock_configuration()</code>	Unlocks all design object versions in the active configuration.
<code>\$\$writeln_monitor()</code>	Writes the specified text string to the configuration's monitor window.

Tool Viewpoint Functions

The tool viewpoint functions are used in qualification and termination scripts for your tool viewpoints.

A qualification script provides the tool viewpoint special start-up instructions. A termination script provides the tool viewpoint with the ability to clean up when you close the tool session. All tool viewpoints *must* have a qualification script by containment or by reference; however, a termination script is optional.

For more information about tool viewpoints, qualification scripts, and termination scripts, refer to the *Pyxis Project Manager User's Manual*.

Table 1-5 describes all tool viewpoint functions.

Table 1-5. Summary of the Tool Viewpoint Functions

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

iDM Interactive Functions

The iDM interactive functions map directly to the **MGC > Design Management** menu selections. The iDM interactive functions provide you with the ability to copy, move, delete, and change the references of design objects, as well as the ability to view the design hierarchy of component design objects from within Mentor Graphics EDA applications.

Although the iDM interactive functions and commands are also available in the Pyxis Project Manager application, whenever possible, you should use the Pyxis Project Manager functions when you are working in the Pyxis Project Manager application. The iDM interactive functions are intended to provide you with design management capabilities from within EDA applications. The Pyxis Project Manager application itself provides you with a complete set of functions and commands needed to manage your data inside it.

Functions that return pathnames always return soft pathnames, if possible. You can determine what hard pathname the soft pathname maps to, by using the `$$get_hard_name()` function. For information on hard, soft, relative, and absolute pathnames, refer to “[Working with References](#)” and “[Glossary](#)” in the *Pyxis Project Manager User's Manual*.

iDM interactive functions are available from within Mentor Graphics EDA applications. For more information about iDM, refer to the *Pyxis Project Manager User's Manual*.

[Table 1-6](#) describes most of the iDM interactive functions. For a specific reference to the Hierarchy window, refer to [Table 6-1](#). For a specific reference to the Component window, refer to [Table 6-2](#).

Table 1-6. Summary of iDM Interactive Functions

Function	Description
\$change_design_object_references()	Changes the references of the specified design objects.
\$copy_design_object()	Copies one or more design objects to a container.
\$delete_design_object()	Deletes the specified design objects.
\$descend_hierarchy_one_level()	Displays the next level of hierarchy directly beneath the selected component in a component hierarchy window.
\$descend_hierarchy_specify_level()	Displays the design hierarchy for a selected component in a component hierarchy window using specified level and filter settings.
\$move_design_object()	Moves one or more design objects to a new container.

iDM Toolkit Functions

The iDM toolkit functions are a set of functions that you use to write scripts to execute from within your EDA application. You can use these scripts to manipulate design objects from within your application in batch mode.

For information about iDM functions you can invoke using menus and the command line, refer to “[iDM Interactive Functions](#)” on page 39.

All of the iDM toolkit functions are available from all Mentor Graphics EDA applications. For more information about iDM, refer to the [Pyxis Project Manager User's Manual](#).

Table 1-7 describes all of the iDM toolkit functions.

Table 1-7. Summary of iDM Toolkit Functions

Function	Description
\$\$change_design_object_references()	Changes the references of the specified design objects.
\$\$copy_design_object()	Copies one or more design objects to a container.
\$\$move_design_object()	Moves one or more design objects to a new container.
\$show_component_hierarchy()	Displays the hierarchy of a design object in a component or IC design hierarchy window.

Language Flow Functions

The language flow functions work on projects, libraries, cells, or HDL source files in the Pyxis Project Manager application.

Table 1-8 lists the language flow functions in the Pyxis Project Manager application.

Table 1-8. Language Flow Functions

Function	Description
\$check_language_views()	Finds invalid language or symbol views inside a given project, library, or component.
\$compile_all_models()	Compiles the currently selected project, library, cell, or HDL source file.
\$create_symbols_for_hdl_lib()	Generates symbols for each design unit in the library, and registers the symbols to their associated models for the given ADMS or Modelsim compiled library path.
\$create_symbols_for_src()	Generates symbols for all design units in the specified source file or hierarchy.
\$get_modules_uniqueness_check_pref()	Returns the current setting value for the Check Module Uniqueness preference as set in the Setup Preferences Dialog Box.
\$import_hdl()	Imports an HDL file, compiles it, and generates symbols for each specified design unit.

Table 1-8. Language Flow Functions

Function	Description
\$is_module_name_unique()	Indicates if a given module name is unique across the compiled library.
\$set_file_compilation_options()	Sets the compilation options for the specified HDL file.
\$set_ini_mappings()	Updates the ADMS INI file for the current hierarchy to include the given set of compiled ADMS libraries.
\$set_modules_uniqueness_check_pref()	Specifies the setting value for the Check Module Uniqueness preference, which indicates whether or not to perform a check for module uniqueness before compilation.
\$set_project_options()	Sets the default compilation options for the currently open hierarchy.
\$set_project_registration_options()	Sets the registration options for the currently open hierarchy.
\$show_invalid_language_views()	Highlights invalid language views and symbols for the specified objects in a set of HDL files.

Chapter 2

Function Dictionary Part 1

This chapter lists the functions available in the Pyxis Project Manager application.

You can find Integrated Design Management functions for the Hierarchy window and the Component window in “[Integrated Pyxis Project Manager Function Dictionary](#)” on page 649.

By default, the menu paths associated with each function apply to the pulldown menus in the GUI. The menu paths from the popup menu, palette menu, or toolbar are called out specifically.

\$\$add_configuration_entry()

Scope: dm_config_tk

Adds the specified design object version to the active configuration.

Usage

```
$$add_configuration_entry("name", "type", version)
```

Arguments

- **name**
A string that specifies the pathname of the design object to be added.
- **type**
A string that specifies the type of the design object to be added.
- **version**
An integer that specifies the version and the state of the design object to be added. If the version is greater than 0, the state of the entry is fixed. If omitted or specified as 0, the state of the entry is current.

The default is 0, which represents the current version.

Return Values

VOID.

Description

The toolkit function `$$add_configuration_entry()` adds the specified version of the design object to the active configuration. This function adds the design object as a primary entry.

If you omit the version argument or specify 0, this function adds the design object version as a “current” entry. A current entry always points to the current version of the design object.

If you specify the version argument as an integer greater than 0, this function adds the specified version of the design object as a “fixed” entry. A fixed entry always points to the same version of the design object.

You add secondary entries, obtained through containment and/or reference traversal, to a configuration by using the function `$build_configuration()`. If the state of the primary entry is fixed, the Pyxis Project Manager application regards all secondary entries referenced by that primary entry as fixed also. The states of the secondary entries are not actually changed to fixed, but are merely interpreted as being overridden for this particular configuration.

Examples

This example opens the “config_a” configuration object and adds version 2 of the “base” design object as a fixed primary entry. Both the configuration object and the new entry are assumed to reside in the project directory `$PROJECT_XYZ`. The “base” design object is of type `Mgc_file`.

```
$$open_configuration("$PROJECT_XYZ/config_a", 0, @read_write);  
$$add_configuration_entry("$PROJECT_XYZ/base", "Mgc_file", 2);
```

Related Topics

[\\$\\$build_configuration\(\)](#)

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

[\\$\\$remove_configuration_entry\(\)](#)

[primary entry](#)

\$add_configuration_entry()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window.

Adds the current version of a design object to the active Configuration window.

Usage

`$add_configuration_entry("object_name", "object_type", update_window)`

ADD COnfiguration Entry object_name object_type *update_window*

Edit > Add Entry

Configuration window popup menu > **Add Entry**

Arguments

- **object_name**
A string that specifies the absolute pathname of the design object to be added.
- **object_type**
A string that specifies the type of the design object to be added.
- ***update_window***
A Boolean that specifies whether to suppress updating of the Configuration window until all entries have been added to the Configuration window. Options include:
 - **@true** — Suppresses updating of the Configuration window until all entries are added. Default.
 - **@false** — Updates the Configuration window after the addition of each entry.

Return Values

VOID.

Description

The interactive function `$add_configuration_entry()` adds the current version of the specified design object to the active Configuration window. This function adds the design object version as a primary entry whose state is "current".

A current entry always points to the current version of the design object. To add a previous version of a design object as a primary entry, you use the `$add_versions()` function.

When you add multiple entries to a configuration during a single execution of the `$add_configuration_entry()` function, you can specify to update the Configuration window after each entry is added or to update the window a single time after all entries are added. You specify the `update_window` argument as `@true` to suppress the update of the Configuration window until all entries are added to the configuration.

Examples

1. This example adds the design object *planet* as a current primary entry to the active Configuration window. The new entry is assumed to reside in the project directory *\$PROJECT_XYZ*. The design object is of type *Mgc_file*.

```
$add_configuration_entry("$PROJECT_XYZ/project/planet", "Mgc_file");
```

2. This example adds the same design object to the active Configuration window by using command syntax from the popup command line.

```
add co e $PROJECT_XYZ/project/planet Mgc_file
```

Related Topics

[\\$add_versions\(\)](#)

[\\$report_entry_info\(\)](#)

[\\$build_configuration\(\)](#)

[\\$report_entry_verification\(\)](#)

[primary entry](#)

[\\$remove_configuration_entry\(\)](#)

[\\$set_build_rules\(\)](#)

\$\$add_container()

Scope: dme_do_tk

Creates a basic container design object with an attribute file.

Usage

```
$$add_container("container_name", "container_type")
```

Arguments

- **container_name**
A string that specifies the pathname of the basic container design object to be added.
- **container_type**
A string that specifies the type of the basic container design object to be added.

Return Values

VOID.

Description

Creates a basic container design object of the type Mgc_container.

A container is a design object with one or more directory fileset members. A basic container design object type has only one fileset member, a directory, and must have the property "basic_container".

This function creates a container that has an attribute file. To create a container that does not have an attribute file, you must use the \$\$add_directory() function.

You must specify the absolute pathname for the container_name argument. If an object with the specified pathname already exists, or if the specified parent container does not exist, this function does not create the container. You specify the container_type argument as Mgc_container.

Examples

This example creates the "tables" container. The container object resides in the \$PROJECT_XYZ project directory.

```
$$add_container("$PROJECT_XYZ/doc/tables", "Mgc_container");
```

Related Topics

[\\$\\$copy_object\(\)](#)

[\\$\\$move_object\(\)](#)

[\\$\\$delete_object\(\)](#)

[\\$\\$add_directory\(\)](#)

\$add_container()

Scope: dmgr_mode

Prerequisite: To use this function, you must be in an active Navigator window.

Creates a basic container design object with an attribute file.

Usage

```
$add_container("container_name", "container_type")
```

ADD COntainer container_name container_type

File > New > Container

Arguments

- **container_name**
A string that specifies either the relative or the absolute pathname of the container to be added.
- **container_type**
A string that specifies the type of the container to be added.

Return Values

VOID.

Description

The interactive function `$add_container()` creates a basic container design object of the `Mgc_container` type. A container is a design object with one or more directory fileset members. A basic container design object type has only one fileset member, a directory and must have the property `"basic_container"`. This function creates a container that has an attribute file. To create a container that does not have an attribute file, you must use the `$add_directory()` function.

The `container_name` argument accepts either an absolute pathname or a relative pathname. If you supply a relative pathname, this function adds the container relative to the working directory of the active navigator. If an object with the specified pathname already exists, or if the specified parent container does not exist, this function does not create the container.

Examples

1. These two examples create a container by using a relative pathname. This function adds the new container relative to the current navigator directory.

```
$add_container("./tables", "Mgc_container");
```

```
$add_container("./tables", "Mgc_container");
```

2. This example creates the container *tables* by using an absolute pathname.

```
$add_container("$PROJECT_XYZ/doc/tables", "Mgc_container");
```

3. This example creates the same container by using command syntax from the popup command line.

```
add co $PROJECT_XYZ/doc/tables Mgc_container
```

Related Topics

[\\$add_directory\(\)](#)

[\\$copy_object\(\)](#)

[\\$delete_object\(\)](#)

[\\$move_object\(\)](#)

\$\$add_directory()

Scope: dme_do_tk

Creates a container design object without an attribute file.

Usage

\$\$add_directory("pathname")

Arguments

- **pathname**

A string that specifies the pathname of the directory to be added.

Return Values

VOID.

Description

The toolkit function \$\$add_directory() creates a container design object without an attribute file, of the Mgc_container type. If a file system object with the specified pathname already exists, or if the specified parent container does not exist, this function does not create the directory.

Examples

The following example creates the “pictures” directory at *\$PROJECT_XYZ/doc*.

```
$$add_directory("$PROJECT_XYZ/doc/pictures");
```

Related Topics

[\\$\\$is_container\(\)](#)

[\\$\\$is_directory\(\)](#)

\$add_directory()

Scope: dmgr_model

Prerequisite: To use this function, you must be in an active Navigator window.

Creates a directory without an attribute file.

Usage

```
$add_directory("directory_name")
```

ADD Directory directory_name

File > New > Directory

Arguments

- **directory_name**

A string that specifies either the relative or absolute pathname of the directory to be added.

Return Values

VOID.

Description

The interactive function `$add_directory()` creates a container design object without an attribute file of the `Mgc_container` type. You can specify either a relative or absolute pathname. If you specify a relative pathname, this function adds the directory relative to the working directory of the active navigator.

If a file system object with the specified pathname already exists, or if the specified parent container does not exist, this function does not create the directory.

Examples

1. The two examples shown here create directories by using a relative pathname. The function adds the new directories relative to the current navigator directory.

```
$add_directory("./pictures");
```

```
$add_directory("../pictures");
```

2. This example creates the directory *tables* by using an absolute pathname.

```
$add_directory("$PROJECT_XYZ/tables");
```

3. This example creates the directory *tables* by using command syntax from the popup command line.

```
add di $PROJECT_XYZ/tables
```

\$add_object_property()

Scope: dmgr_mode

Prerequisite: To use this function, you must be in an active Navigator window and must select at least one design object.

Adds a property to the selected design object.

Usage

```
$add_object_property("prop_name", "prop_value")
```

```
ADD OBJECT Property prop_name prop_value
```

File > Add Property

Arguments

- **prop_name**
A string that specifies the name of the property to be added. This property name can contain blank spaces.
- **prop_value**
A string that specifies the value of the property to be added.

Return Values

VOID.

Description

The interactive function \$add_object_property() adds the specified object property to the current version of the selected design object. Adding a property does not create a new version of the object. As new versions of the design object are created, the object properties remain associated with the current version of the design object. To view the object properties, use the \$report_object_info() function.

Before you can add a property to the selected design object, you must have permission to edit the object. You can use the \$change_protection() function to change the protection status of the object.

Examples

1. This example adds the property "test_number" to the selected design object, and assigns the value "1A" to the property.

```
$add_object_property("test_number", "1A");
```

2. This example adds the same property by using command syntax from the popup command line.

```
add ob p test_number 1A
```

Related Topics

[\\$add_object_property\(\)](#)

[Design Object Properties](#)

[\\$change_object_property\(\)](#)

[\\$delete_object_property\(\)](#)

[\\$change_protection\(\)](#)

[\\$report_object_info\(\)](#)

\$add_project_to_rc()

Scope: dmgr_project_model

Adds the specified root hierarchy to the specified server and repository.

Usage

`$add_project_to_rc("server_name", "repository", "root_hdo", comment)`

Arguments

- **server_name**
A string that specifies the name of the server to use when adding the project.
- **repository**
A string that specifies the name of the repository to use when adding the project.
- **root_hdo**
A string that specifies the path of the project to add.
- ***comment***
A string that specifies the comment to use when adding the project. The default is empty string.

Return Values

VOID.

Description

If you omit the comment argument, this function uses a default comment of "\$TYPE found at \$PATH added to the repository."

This function need only be called once per root hierarchy. Each root hierarchy in a repository must have a distinct name.

Examples

```
$add_project_to_rc($rcs_default_server(), $rcs_default_repository(), "$PROJECT");  
  
$add_project_to_rc($rcs_default_server(), $rcs_default_repository(), "$PROJECT", \  
"Adding PROJECT to repository");
```

Related Topics

[\\$create_work_area_from_rc\(\)](#)

[\\$checkin_objects_to_rc\(\)](#)

[\\$update_objects_from_rc\(\)](#)

[\\$cancel_checkout_objects_for_rc\(\)](#)

[\\$checkin_objects_to_rc\(\)](#)

[\\$object_status_for_rc\(\)](#)

[\\$revert_object_from_rc\(\)](#)

\$\$add_reference()

Scope: dme_do_tk

Creates a reference from the specified source design object to the specified target design object.

Usage

```
$$add_reference("obj_name", "obj_type", obj_version, "target_obj_name", "target_type",  
               target_version)
```

File > Add Reference

Arguments

- **obj_name**
A string that specifies the pathname of the source design object.
- **obj_type**
A string that specifies the type of the source design object.
- **obj_version**
An integer that specifies the version number of the source design object. The default is the current version.
- **target_obj_name**
A string that specifies the pathname of the target design object.
- **target_type**
A string that specifies the type of the target design object.
- **target_version**
An integer that specifies the version and the *reference state* of the target design object. If the version is greater than 0, the state of the reference is *fixed*. If omitted or specified as 0, the state of the reference is *current*. By default, the state of the reference is current.

Return Values

An integer that specifies the reference handle. The reference handle uniquely identifies the reference. If an error occurs, this function returns VOID.

Description

A reference is unidirectional; it always points from the source design object to the target design object.

To create a reference from a previous version of the source design object, you must specify the *object_version* argument; otherwise, this function adds the reference to the current version of the source design object.

If you omit the `target_version` argument or specify 0, this function adds the reference as a “current” reference. A current reference always points to the current version of the design object.

If you specify the `target_version` argument as an integer greater than 0, this function adds the reference as a “fixed” reference. A fixed reference always points to the design object version to which it was originally set. If you specify the current version number for the `target_version` argument, the reference is added as a fixed reference that points to the current version. As the design object evolves and its version number is incremented, the fixed reference remains with the version to which it was originally set. Adding a reference to a previous version of the target design object has no effect on other versions of the design object.

If you specify a reference whose name, type, state, and version match an existing reference, this function does not create a duplicate reference; instead, it returns the handle of the existing reference. It is possible for other Mentor Graphics Pyxis applications, such as Pyxis Schematic, to create duplicate references.

When you add a reference to a design object, this function attaches a reference property “creating_tool” whose value is “project manager”. The design object toolkit functions can only edit references that hold this property.

Before you can add a reference, you must have permission to edit the source design object and permission to read the target design object. You can use the `$change_protection()` function to change the protection status of design objects. To change a design object's references, you use the configuration management toolkit function `$change_object_references()`.

Examples

1. This example adds a reference from the “library” source design object to version 2 of the “base” target design object. As the `target_version` argument *is* specified, the state of this reference is fixed.

```
$$add_reference("$PROJECT_XYZ/library", "Mgc_container", ,\
"$PROJECT_ABC/d1/base", "Document_file", 2);
```

2. This example adds a reference from the “library” source design object to the “base” target design object. As the `target_version` argument *is not* specified, the state of the reference is current.

```
$$add_reference("$PROJECT_XYZ/library", "Mgc_container", ,\
"$PROJECT_ABC/d1/base", "Document_file", );
```

As the reference in example 1 is fixed and the reference in example 2 is current, they are not duplicate references. Each reference returns a different target handle.

Related Topics

[\\$\\$change_object_references\(\)](#)

[\\$\\$get_object_references\(\)](#)

[\\$\\$delete_reference\(\)](#)

[\\$\\$get_reference_properties\(\)](#)

\$\$delete_reference_property()

\$\$has_reference_property()

\$\$delete_reference_handle()

\$\$set_reference_property_handle()

\$change_protection()

\$add_reference()

Scope: dmgr_mode

Prerequisite: To use this function, you must be in an active Navigator window and must select at least one design object.

Creates a reference from the selected design object to a target design object.

Usage

```
$add_reference("ref_obj_name", "ref_obj_type")
```

ADD REference ref_obj_name ref_obj_type

File > Add Reference

Arguments

- **ref_obj_name**
A string that specifies the name of the target design object.
- **ref_obj_type**
A string that specifies the type of the target design object.

Return Values

VOID.

Description

A reference is unidirectional; it only points from the source design object to the target design object.

Adding a reference does not create a new version of the selected design object. This function adds the reference to the current version of the selected design object, and the state of the reference is *current*. A current reference always points to the current version of the target design object. A *fixed* reference points to a design object version other than the current version. To change the state of a reference, use the `$change_reference_state()` function.

Although the `ref_obj_type` argument is required, you can avoid supplying the design object type by using the menu interface. When you execute this function via one of the menu paths, a dialog navigator displays enabling you to specify the new reference's target object by clicking on the desired object. When you select the reference's target object by using the dialog navigator, the design object's type is automatically supplied.

When you add a reference to a design object, this function attaches a reference property "creating_tool" whose value is "project manager". Only the Pyxis Project Manager application can edit or delete references that hold this reference property. For information on Pyxis Project Manager properties, refer to the *Pyxis Project Manager User's Manual*.

Before you can add a reference, you must have permission to write the source design object and permission to read the target design object. You can use the `$change_protection()` function to change the protection status of design objects.

Examples

1. This example adds a current reference from the selected design object to the target design object *canvas*, which is in the same directory.

```
$add_reference("canvas", "Mgc_file");
```

2. This example adds a current reference from the selected design object to the target design object *canvas* in the directory *\$PROJECT_XYZ/design*.

```
$add_reference("$PROJECT_XYZ/design/canvas", "Mgc_file");
```

3. This example adds the same reference by using command syntax from the popup command line.

```
add re $PROJECT_XYZ/design/canvas Mgc_file
```

Related Topics

[\\$change_reference_state\(\)](#)

[\\$delete_reference\(\)](#)

[\\$change_protection\(\)](#)

[\\$report_reference_info\(\)](#)

[\\$show_references\(\)](#)

[References](#)

\$add_reference_property()

Scope: dmgr_reference_mode

Prerequisite: To use this function, you must be in an active Reference window and must select at least one reference.

Adds a property to the selected reference.

Usage

```
$add_reference_property("prop_name", "prop_value")
```

```
ADD REference Property prop_name prop_value
```

Add > Reference Property

Arguments

- **prop_name**
A string that specifies the name of the property to be added. This string can contain blank spaces.
- **prop_value**
A string that specifies the value of the property to be added. The default is "", which is an empty string.

Return Values

VOID.

Description

Adding a reference property does not create a new version of the source design object. This function adds the reference property to the current version of the source design object. As new versions of the design object are created, the reference property remains associated with the current version of the source design object.

To view the reference property, you can use the \$report_reference_info() function. To add a reference property, you must have permission to edit the selected source design object. You can use the \$change_protection() function to change the protection status of design objects.

Examples

1. This example adds the property "test_results" and the value "Pass" to the selected reference.

```
$add_reference_property("test_results", "Pass");
```

2. This example adds the property "test_results" by using command syntax from the popup command line.

```
add re p test_results Pass
```

Related Topics

[\\$change_reference_property\(\)](#)

[\\$\\$delete_reference_property\(\)](#)

[\\$select_reference\(\)](#)

[\\$report_reference_info\(\)](#)

\$add_toolbox()

Scope: dmgr_toolbox_area

Prerequisite: To use this function, you must be in an active Tools window that is in toolbox mode.

Appends the specified toolbox to the current toolbox search path.

Usage

`$add_toolbox("pathname")`

ADD TOolbox pathname

Edit > Add Toolbox

Toolboxes window > Add Toolbox

Arguments

- **pathname**

A string that specifies the absolute pathname of the toolbox to be added.

Return Values

VOID.

Description

The [toolbox search path](#) is the lower right corner of the window. The new toolbox directory must exist to be added to the toolbox search path.

When you specify the toolbox pathname argument, you can specify either an absolute or a relative pathname. If you specify a relative pathname, the pathname is considered relative to the working directory. The Pyxis Project Manager application does not enable multiple toolboxes, with either the same or different pathnames, to point to the same directory.

Examples

1. This example adds the toolbox *testbox3* to the toolbox search path.

```
$add_toolbox("$PROJECT_D/support/testbox3");
```

2. This example adds the same toolbox by using command syntax from the popup command line.

```
add tool $PROJECT_D/support/testbox3
```

Related Topics

[\\$open_tools_window\(\)](#)

[\\$view_tools\(\)](#)

[\\$remove_toolbox\(\)](#)

[\\$view_toolboxes\(\)](#)

`$save_toolbox_search_path()`

Toolboxes

\$\$add_type()

Scope: dmgr_proj_tk

Adds a new type to the custom type registry.

Usage

```
$$add_type("type_name", "tool_name", "default_tool", "icon_string", "large_icon_string",  
           "key_name", "key_type")
```

Setup > New Type

Arguments

- **type_name**
A string specifying the name of the type. The name should not match any existing type names.
- **tool_name**
An optional string specifying the name of the default tool for opening objects of this type.
- **default_tool**
An optional string specifying the path to the default tool for opening objects of this type.
- **icon_string**
An optional string specifying the path to the file to be used as a small icon for this type.
- **large_icon_string**
An optional string specifying the path to the file to be used as a large icon for this type.
- **key_name**
An optional string specifying the extension that is used to recognize objects of this type.
- **key_type**
An optional string specifying whether this type is a file type or a directory type. If the key_type string is "File", then files with the extension specified by key_name are recognized as objects of this type. If the key_type string is "Directory", then directories with the extension specified by key_name are recognized as objects of this type.

Return Values

VOID.

Description

Sets the file extension used to recognize the new type. Sets icons for displaying objects of the new type and a default tool for opening those objects. This function requires that the user has first logged in as an administrator using \$login_admin().

Examples

The following example adds a new type “new_custom_txt”. The type is recognized by the file extension *.txt*. Objects of this type are displayed with the small icon at */tmp/txt_icon*, and do not have a large icon. The custom_txt objects opens with */usr/bin/xemacs*.

```
$add_type("new_custom_txt", "", "/usr/bin/xemacs", "/tmp/txt_icon", "", "txt", "File");
```

Related Topics

[\\$login_admin\(\)](#)

[\\$\\$update_type\(\)](#)

\$add_versions()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window and must select only one configuration entry.

Adds the specified versions of the selected design object to the active Configuration window.

Usage

`$add_versions([version_number])`

ADD VErSIONS [version_number]

Edit > Add Versions

Configuration > Add Versions

Arguments

- **version_number**

A vector that lists the design object version numbers to be added.

Return Values

VOID.

Description

The interactive function `$add_versions()` adds the specified versions of the selected design object to the active Configuration window. This function adds the new version of both primary and secondary entries as fixed primary entries. A fixed entry always points to a version other than the current version of the design object that it references, unless the object has not evolved to another version since the entry was fixed.

Versions that you add from both primary and secondary entries inherit the build rules.

Examples

1. This example adds versions 2 and 3 of the selected design object to the Configuration window.

`$add_versions([2, 3]);`

2. This example adds the same versions by using command syntax from the popup command line.

`add ve [2, 3]`

Related Topics

[\\$add_configuration_entry\(\)](#)

[\\$remove_configuration_entry\(\)](#)

`$build_configuration()`

`$set_build_rules()`

Versions

\$browse_for_object()

Scope: dmgr_session_window

Prerequisite: You can use this function from any Pyxis Project Manager window.

Brings up an object browser dialog box.

Usage

`$browse_for_object([filevec])`

BROwse FOr Object [filevec]

File > New or **File > Open**

Arguments

- **filevec**

This vector receives the output vector produced by the dialog navigator. For more information, refer to the usage of the `$dialog_navigator()` function in the [\\$set_form_gadget_value\(\)](#) example of the *Pyxis Common User Interface Reference Manual*.

Return Values

A string that specifies the absolute pathname of the design object selected in the dialog navigator.

Description

The interactive function `$browse_for_object()` brings up a dialog navigator. The dialog navigator is a dialog box that enables you to navigate to the design objects in your file system, and to select design objects on which to work. The dialog navigator accepts both hard and soft pathnames as input, and displays soft pathnames, if possible.

This function maps to the Ctrl-MenuBar key combination. You can use the Ctrl-MenuBar key combination to invoke the dialog navigator from a “pathname” field in a prompt bar, and then use the dialog navigator to locate and select the design object whose pathname you want to enter in the prompt bar. After you select the design object and click the **OK** button, the dialog navigator enters the soft path equivalent of the object you selected, into the pathname field of the active prompt bar.

You can specify which design objects are displayed in the dialog navigator list area by clicking on the dialog navigator Filter button and specifying UNIX System V wildcards.

You can also use this function to return a design object's pathname to the message area of the session, as shown in example 2.

Examples

1. To browse for a design object when a function brings up a prompt bar, you press the Ctrl-MenuBar key combination.

2. This example returns the design object's pathname to the message area by using function syntax from the popup command line:

`$message($browse_for_object());`

3. This example brings up the dialog navigator by using command syntax from the popup command line.

`bro fo o`

Related Topics

[`\$search\(\)`](#)

[`\$set_form_gadget_value\(\)`](#)

\$\$build_configuration()

Scope: dm_config_tk

Adds secondary entries to the active configuration, based on the build rules of the primary entries.

Usage

\$\$build_configuration()

Configuration window toolbar > **Build Configuration**

Configuration window popup menu > **Build**

Arguments

None.

Return Values

VOID.

Description

The toolkit function \$\$build_configuration() adds secondary entries to the active configuration, based on the build rules of the primary entries. By setting the build rules of a primary entry, you can include or exclude secondary entries from a configuration based on the design object's pathname, type, object properties, and reference properties. You can also turn off reference traversal.

The state of an included secondary entry depends on the state of the primary entry from which it is obtained. The state of an entry may be either current or fixed. A current entry always points to the current version of the design object that it references. A fixed entry always points to the same version of the design object that it references.

The states of secondary entries obtained by the \$\$build_configuration() function are determined as follows:

- Secondary entries obtained through the containment hierarchy are always current entries.
- Secondary entries obtained through references are current, unless one of the following conditions are true:
 - The reference through which entries are obtained is fixed.
 - The design object that holds the reference is released.
 - The entry for the design object that holds the references is fixed.

If you do not specify build rules for a primary entry, the primary entry inherits the default build rules, which include all contained design objects and all referenced design objects.

The `$$build_configuration()` function automatically resolves any target conflicts that might exist in the configuration. For each conflicting retargetable entry, `$$build_configuration()` includes the entry's parent container. If the addition of the parent entry creates a new conflict, the parent's parent container is added, until all conflicts are resolved.

**Note**

For configurations of “Dme_config_do” type, add the “creating_tool” reference property whose value equals “Dme_config”, to the configuration object. This reference property is always present and indicates that the reference was created by the configuration object and should not be changed or deleted by the Pyxis Project Manager application, or any other tool.

A second reference property, “Dme_config_ignore” whose value is “value__void”, is placed on certain references to prohibit the `$$build_configuration()` function from traversing references that are added by the configuration object, but that do not correspond to actual configuration entries.

Examples

The following example opens the “config_beta” configuration object, adds a fixed entry, and then builds the configuration by using the default build rules.

```
$$open_configuration("$PROJECT_XYZ/d1/config_beta", 0,  
    @read_write); $$add_configuration_entry("$PROJECT_XYZ/d2/example/sheet1",  
    "Dss_sheet", 1);  
  
$$build_configuration();
```

Related Topics

[\\$\\$add_configuration_entry\(\)](#)[\\$\\$remove_configuration_entry\(\)](#)[\\$\\$set_object_path_filter\(\)](#)[\\$\\$set_object_property_filter\(\)](#)[\\$\\$set_object_type_filter\(\)](#)[\\$\\$set_reference_property_filter\(\)](#)[\\$\\$set_reference_traversal\(\)](#)[\\$\\$set_target_path\(\)](#)

\$build_configuration()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window.

Adds secondary entries to the active Configuration window, based on the build rules of the primary entries.

Usage

\$build_configuration()

BUild COnfiguration

Arguments

None.

Return Values

VOID.

Description

By setting the build rules of a primary entry, you can specify which design objects to include in the configuration and which design objects to exclude from the configuration.

The \$build_configuration() function automatically resolves any target conflicts that might exist in the configuration. For each conflicting retargetable entry, \$build_configuration() includes the entry's parent container. If the addition of the parent entry creates a new conflict, the parent's parent container is added until all conflicts are resolved.

The \$build_configuration() function writes a real-time status summary to the configuration's monitor window as it executes. The summary includes the number of failures, errors, and warnings found during the build operation. A warning is reported during a build operation when a current reference fails to point to the current version of the design object it references. Additionally, a message is written to the message area, stating whether the build operation failed, passed, or passed with warnings.

If your session setup values specify to show the configuration monitor window, the monitor window is opened when you execute this function. When the operation is complete, the monitor window remains visible until you return to the Configuration window by executing the **Hide Monitor** popup menu item. You can redisplay the operation's monitor window summary by executing the Configuration window's popup menu item **Show Monitor**.

The configuration's monitor window is cleared when you execute a new configuration operation. To save the status information in a file, you can use the configuration monitor window popup menu item **Export**.

Building a configuration by using the \$build_configuration() function is an interruptible operation. At any time during the execution of this function, you can press the Kill key to interrupt the operation. When you halt the build operation, a dialog box appears prompting you to abort or to continue the operation. If you choose to abort, the operation is halted.

Interrupting the build operation, in this way, halts the operation, but the Pyxis Project Manager application does not return the configuration to its original state. The new secondary entries, which were added to the active Configuration window prior to the interrupt, are now visible in the active Configuration window.

If you press the Kill key from outside the scope of a configuration operation, you are queried on whether you want to close the Pyxis Project Manager session. By selecting the **Yes** button, you exit from the Pyxis Project Manager application. By selecting **No**, you return to the active Pyxis Project Manager window from which you entered the command.

The `$set_build_rules()` function enables you to set the build rules of a primary entry. You can set the pathname, type, and object property filters, and turn reference traversal to *on* or *off*.

The state of an included secondary entry depends on the build rules of the primary entry from which it is obtained. The state of an entry may be either current or fixed. A current entry always points to the current version of the design object that it references. A fixed entry always points to a version other than the current version of the design object that it references.

The states of secondary entries obtained by the `$build_configuration()` function are determined as follows:

- Secondary entries obtained by containment are always current entries.
- Secondary entries obtained through references are current, unless the one of the following conditions are true:
 - The reference through which entries are obtained is fixed.
 - The source design object of the reference is released.
 - The entry for the design object that holds the reference is fixed.

Note that if you do not specify build rules for a primary entry, the primary entry inherits the default build rules, which include all contained design objects and all referenced design objects.

Examples

1. This example builds the configuration in the active Configuration window by using the default build rules.

```
$build_configuration();
```

2. This example builds the same configuration by using command syntax from the popup command line.

```
bui co
```

Related Topics

[\\$add_configuration_entry\(\)](#)

[\\$report_configuration_info\(\)](#)

[\\$open_configuration_window\(\)](#)

[\\$set_build_rules\(\)](#)

[\\$remove_configuration_entry\(\)](#)

[View Primary and Secondary Entries](#)

\$cancel_checkout_objects_for_rc()

Scope: dmgr_project_model

Cancels the checkout for the specified objects and their children.

Usage

`$cancel_checkout_objects_for_rc([objects], recurse)`

Arguments

- **objects**

A vector containing a list of objects to execute with cancel checkout. Each object is itself a vector containing exactly the following two strings:

- The first string is the object pathname.
- The second string is the object type.

- ***recurse***

A Boolean that specifies whether cancel checkout occurs hierarchically.

Return Values

VOID.

Description

This function requires that all objects reside in a managed hierarchy. See `$add_project_to_rc()`. Relinquishes the lock the caller has on the object, enabling other users to check out the objects.

For objects that have pending edits, this function reverts those objects with the latest version in the repository.

Objects that are not checked out by the current user are skipped. Objects that are unmanaged are also skipped.

Examples

```
$cancel_checkout_objects_for_rc([[ "$PROJECT/lib/cell", "mgc_component" ]]);
```

```
$cancel_checkout_objects_for_rc([[ "$PROJECT/lib/cell", "mgc_component" ], \
[ "$PROJECT/lib/cat", "mgc_category" ]]);
```

Related Topics

[\\$add_project_to_rc\(\)](#)

[\\$create_work_area_from_rc\(\)](#)

[\\$checkin_objects_to_rc\(\)](#)

[\\$checkout_objects_from_rc\(\)](#)

[\\$update_objects_from_rc\(\)](#)

[\\$object_status_for_rc\(\)](#)

[\\$revert_object_from_rc\(\)](#)

\$\$change_configuration_references()

Scope: dm_config_tk

Changes the references of each entry in the active configuration.

Usage

`$$change_configuration_references([pattern_list], preview_mode, lock_mode)`

Arguments

- **pattern_list**

A vector of strings that specifies the pattern to be matched and the string with which to replace the pattern. The format of this vector is:

```
[“from_pathname_pattern1”, “to_pathname1”, “from_pathname_pattern2”,  
 “to_pathname2”, ..., “from_pathname_patternN”, “to_pathnameN”]
```

The `from_pathname_pattern` is the string pattern to be replaced and the `to_pathname` is the string pattern to substitute in its place. You can specify the `from_pathname_pattern` string by entering a literal string or by using System V regular expressions.

- **preview_mode**

A name that indicates whether the operation actually executes and modifies the entries' references, or only reports the expected results of an actual execution, without performing the changes.

Choose one of the following:

- **@nopreview (-Nopreview)** — The operation is performed. Default.
- **@preview (-Preview)** — The results of the operation are reported to the configuration status report area, without the operation being performed.

- **lock_mode**

A name that specifies whether the configuration entry is locked before this function attempts to change its references. This argument can be set to either `@nolock` or `@lock`.

Choose one of the following options:

- **@nolock (-Nolock)** — The configuration entry is not locked before this function attempts to change its references.
- **@lock (-Lock)** — The configuration entry is locked before this function attempts to change its references. Default.

Return Values

VOID.

Description

The toolkit function `$$change_configuration_references()` changes the reference pathnames of each design object version in the active configuration by matching and replacing each pair of strings specified in the `pattern_list` argument. If the `@preview` switch is specified, the reference pathnames are not actually changed, but the expected affects of the specified changes are reported.

Depending upon which configuration is targeted by this function, the preview report can be viewed in either the configuration monitor window, by using the Configuration window's popup menu item **Show Monitor**, or in the session configuration monitor window, by using the pulldown menu item **Windows > Open Session Monitor**.

By default, configuration status is reported to the active configuration's monitor window. If the session defaults have been modified to report session configuration status to the transcript window, the status report can be viewed by executing the pulldown menu item **MGC > Transcript > Show Transcript**.

This function changes the references of design objects, as they are stored in the reference. Automatic soft-to-hard or hard-to-soft pathname conversion on either the pattern you enter or the stored reference pathname is not performed.

This function and the `$$change_object_references()` function can operate on references that are created by other tools. All other Pyxis Project Manager functions can only operate on design objects with the property “creating_tool”, whose value is equal to “project manager”.

Examples

The following example opens the “config_a” configuration and changes each configuration entry with the “design1” reference pathname pattern to the new “design2” reference pathname pattern and each configuration entry with the “and1” reference pathname pattern to the new “and2” reference pathname pattern. A regular expression (^) denotes that the pattern must occur at the beginning of the string. If a reference contains more than one matching string, only the first match is changed. Therefore, if config_a has a reference that matches both “^/design1” and “and1”, then /design1 is replaced with /design2 and “and1” remains unchanged.

```
$$open_configuration("$PROJECT_XYZ/config_a", 0, @read_write);
```

```
$$change_configuration_references(["^/design1", "/design2", \  
    "and1", "and2"], @npreview, @lock);
```

Related Topics

[\\$\\$change_object_references\(\)](#)

\$change_configuration_references()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window.

Changes the references of each entry in the active Configuration window.

Usage

`$change_configuration_references([pattern_list], preview_mode, lock_mode)`

CHAnge COnfiguration References [*pattern_list*] *preview_mode lock_mode*

Edit > Change > References

Arguments

- **pattern_list**

A vector of strings that specifies the pattern to be matched and the string with which to replace the pattern. The format of this vector is:

```
[“from_pathname_pattern1”, “to_pathname1”, “from_pathname_pattern2”,  
  “to_pathname2”, ..., “from_pathname_patternN”, “to_pathnameN”]
```

The `from_pathname_pattern` is the string pattern to be replaced and the `to_pathname` is the string pattern to substitute in its place. You can specify the `from_pathname_pattern` string by entering a literal string or by using System V regular expressions.

- **preview_mode**

A name that indicates whether the operation actually executes and modifies the entries' references, or only reports the expected results of an actual execution, without performing the changes.

Choose one of the following:

- **@nopreview (-Nopreview)** — The operation is performed. Default.
- **@preview (-Preview)** — The results of the operation are reported to the configuration status report area, without the operation being performed.

- **lock_mode**

A name that specifies whether the configuration entry is locked before this function attempts to change its references. This argument can be set to either `@nolock` or `@lock`.

Choose one of the following options:

- **@nolock (-Nolock)** — The configuration entry is not locked before this function attempts to change its references.
- **@lock (-Lock)** — The configuration entry is locked before this function attempts to change its references. Default.

Return Values

VOID.

Description

The interactive function `$change_configuration_references()` function changes the reference pathnames of each design object version in the active Configuration window by matching and replacing each pair of strings specified in the `pattern_list` argument. If the `@preview` switch is specified, the reference pathnames are not actually changed, but the effects of the specified changes are reported in the configuration's monitor window.

This function changes the references of objects, as they are stored in the reference. Automatic soft-to-hard or hard-to-soft pathname conversion on either the pattern you enter or the stored reference pathname is not performed.

Changing the references of a configuration with the `$change_configuration_references()` function is an interruptible operation. At any time during its execution, you can press the Kill key to interrupt the operation.

When you halt this operation, a dialog box is displayed, prompting you to abort or to continue the operation. If you choose to abort, the operation is halted. Interrupting the operation in this way, halts the operation, but the Pyxis Project Manager application does not return the configuration to its original state. Any changes made to the reference pathnames of an entry prior to the interruption are permanent, while those reference pathnames that have not been processed prior to the interruption remain unaffected.

While executing, the `$change_configuration_references()` function writes a status report to either the configuration's monitor window or to the transcript window. The summary includes the number of failures, errors, and warnings found during the operation. Additionally, a message appears in the message area, stating whether the operation failed, passed, or passed with warnings.

If your session setup values specify to display monitoring status in the monitor window and to display the monitor window, the monitor window is opened when you run this function. When the operation is complete, the monitor window remains visible until you return to the Configuration window by executing the monitor window's popup menu item **Hide Monitor**. You can redisplay the operation's status summary by executing the Configuration window's popup menu item **Show Monitor**.

The configuration's monitor window is cleared upon the execution of a new configuration operation. To save the status information in a file, you can use the Configuration window popup menu item **Export**.

This function and the `$change_object_references()` function can operate on references that are created by other tools. All other Pyxis Project Manager functions can only operate on design objects with the property "creating_tool", whose value is "project manager".

Examples

1. This example changes each configuration entry with the “design1” reference pathname pattern to the new “design2” reference pathname pattern. A regular expression (^) denotes that the pattern must be found at the beginning of the string.

```
$change_configuration_references(["^design1","design2"], \  
@nopreview);
```

2. This example changes configuration references by using command syntax from the popup command line.

```
cha co r ["^design1/base", "design1/area"]
```

Related Topics

[\\$add_configuration_entry\(\)](#)

[\\$report_configuration_references\(\)](#)

[\\$remove_configuration_entry\(\)](#)

[\\$change_object_references\(\)](#)

\$\$change_design_object_references()

Scope: `dme_base_tk`

Changes the references of the specified design objects.

Usage

`$$change_design_object_references([objects], [from_to], lock)`

Arguments

- **objects**

A vector of string vectors that specifies the design objects whose references are to be changed. The format of the vector of string vectors is:

`[["object1", "type1"], ["object2", "type2"], ..., ["objectN", "typeN"]]`

The object is a string that specifies the absolute or relative pathname of the design object. The type is a string that specifies the type of the design object.

- **from_to**

A vector of strings that specifies the pattern to be matched and the string with which to replace the pattern. The format of this vector is:

`["from_pathname_pattern1", "to_pathname1", "from_pathname_pattern2",
"to_pathname2", ..., "from_pathname_patternN", "to_pathnameN"]`

The `from_pathname_pattern` is the string pattern to be replaced and the `to_pathname` is the string pattern to substitute in its place. You can specify the `from_pathname_pattern` string by entering a literal string or by using System V regular expressions.

- **lock**

A name that specifies whether the design object is locked before this function attempts to change its references. This argument can be set to either `@nolock` or `@lock`. The default is either the last option specified or, if you have not executed this function in the current session, `@lock`.

Choose one of the following options:

- **@nolock (-Nolock)** — The design object is not locked before this function attempts to change its references.
- **@lock (-Lock)** — The design object is locked before this function attempts to change its references.

Return Values

An integer that indicates if the design object references were successfully changed. If the design object references were successfully changed, this function returns a 0 for successful completion; otherwise, this function returns a 1 for error.

Description

The iDM toolkit function `$$change_design_object_references()` changes the references of all versions of each design object specified in the `objects` argument. This function replaces the string matched in the `from_pathname_pattern` portion of the `from_to` argument with the string in the `to_pathname` portion of the `from_to` argument.

For each reference, this function attempts to match each of the `from_pathname_pattern` argument until a successful match is found. When a successful match is found, this function replaces the matched pattern with the corresponding `to_pathname_pattern` argument and then proceeds to the next reference. This function does not search for multiple pattern matches in each reference; it only performs pattern matching and replacement for the first matched pattern in a reference.

This function changes the references of design objects, as they are stored in the reference. If you specify a hard pathname replacement pattern, the specified hard pathname is not converted to a soft pathname, but is stored exactly as you specified it.

If you execute this function and specify the `@nolock` option, the specified design object is not locked before its references are changed. Although specifying the `@nolock` option enables this function to execute faster, you should only use this option when you are absolutely sure that the target design object is not changed by another user during the function's execution. If the target design object is modified while this function is executing, your data can be corrupted.

The `$$change_design_object_references()` function can operate on references that are created by tools other than the Pyxis Project Manager application; that is, they can operate on the references of design objects that do not have the property “`creating_tool`” with the value “`project manager`”.

Examples

1. This example changes all the references in all versions of the “`add_det`” component whose pathnames match the patterns “`^$PROJ_X`” or “`and`”. After locking the `add_det` component, this function replaces occurrences of the `^$PROJ_X` pattern with the `$PROJ_Y` pattern and occurrences of the “`and`” pattern with the “`and2`” pattern. A regular expression (^) denotes that the pattern must occur at the beginning of the string.

If a reference contains more than one matching string, only the first match is changed. As shown below, if the `add_det` component has a reference that matches both the `^PROJ_X` and “`and`” patterns, then `PROJ_X` is replaced with `PROJ_Y` but “`and`” remains unchanged.

```
$$change_design_object_references(
    [["$PROJECT_XYZ/ch_dir/add_det", "mgc_component"]],
    ["^$PROJ_X", "$PROJ_Y", "and", "and2"], @lock);
```

References of `add_det` before executing the function:

```
$PROJ_X/chref_dir/b1
$PROJ_X/design1/and
$PROJ_Z/design2/and/and
```

References of `add_det` after executing the function:

```
$PROJ_Y/chref_dir/b1  
$PROJ_Y/design1/and  
$PROJ_Z/design2/and2/and
```

2. This example changes all the references in all versions of the “card_reader” component that have the pattern “/idea/user/dpm_rlslib” in their pathname. Without locking the `card_reader` component, this function replaces each occurrence of the `/idea/user/dpm_rlslib` pattern with the soft prefix `$DPM_DPMRLSLIB`.

```
$$change_design_object_references(  
  [["$PROJECT_ABC/test_dir/card_reader", "mgc_component"]],  
  ["/idea/user/dpm_rlslib", "$DPM_DPMRLSLIB"], @nolock);
```

Related Topics

[`\$change_design_object_references\(\)`](#)

\$change_design_object_references()

Scope: `dme_base_tk`

Prerequisite: To use this function, you must be in an active Navigator window.

Changes the references of the specified design objects.

Usage

`$change_design_object_references([objects], [from_to], lock)`

CHAnge DESign Object References `[objects] [from_to] lock`

Edit > Change > References

Arguments

- **objects**

A vector of string vectors that specifies the design objects whose references are to be changed. The format of the vector of string vectors is:

`[["object1", "type1"], ["object2", "type2"], ..., ["objectN", "typeN"]]`

The object is a string that specifies the absolute or relative pathname of the design object. The type is a string that specifies the type of the design object. The default is set to whatever is selected.

- **from_to**

A vector of strings that specifies the pattern to be matched and the string with which to replace the pattern. The format of this vector is:

`["from_pathname_pattern1", "to_pathname1", "from_pathname_pattern2",
"to_pathname2", ..., "from_pathname_patternN", "to_pathnameN"]`

The `from_pathname_pattern` is the string pattern to be replaced and the `to_pathname` that follows it is the string pattern to substitute in its place. You can specify the `from_pathname_pattern` string by entering a literal string or by using System V regular expressions.

- **lock**

A name that specifies whether the design object is locked before this function attempts to change its references. This argument can be set to either `@nolock` or `@lock`. The default is either the last option specified or, if you have not executed this function in the current session, `@lock`.

Choose one of the following options:

- **@nolock (-Nolock)** — The source design object is not locked before this function attempts to change its references.
- **@lock (-Lock)** — The source design object is locked before this function attempts to change its references.

Return Values

VOID.

Description

The iDM interactive function `$change_design_object_references()` changes the references of all versions of each design object specified in the `objects` argument. This function replaces the string matched in the `from_pathname_pattern` portion of the `from_to` argument with the string in the `to_pathname` portion of the `from_to` argument.

For each reference, this function attempts to match each of the `from_pathname_pattern` arguments until a successful match is found. When a successful match is found, this function replaces the matched pattern with the corresponding `to_pathname` argument and then proceeds to the next reference. This function does not search for multiple pattern matches in each reference; it only performs pattern matching and replacement for the first matched pattern in a reference.

This function changes the references of design objects, as they are stored in the reference. If you specify a hard pathname replacement pattern, the specified hard pathname is not converted to a soft pathname, but is stored exactly as you specified it.

If you execute this function and specify the `@nolock` option, the source design object is not locked before its references are changed. Although specifying the `@nolock` option enables this function to execute faster, you should only use this option when you are absolutely sure that the source design object will not be changed during the function's execution. If the source design object is modified while this function is executing, your data can be corrupted.

The `$change_design_object_references()` function can operate on references that are created by tools other than the Pyxis Project Manager application; that is, they can operate on the references of design objects that do not have the property "creating_tool" with the value "project manager".

When you execute the `$change_design_object_references()` function using either the menu interface or command syntax, the Change Design Object References dialog box displays enabling you to easily enter your argument selections.

The Change Design Object References dialog box contains a dialog navigator with which you can select the design objects whose references you want to change, a filter option which enables you to limit the set of design objects you see displayed in the dialog navigator, a lock option which enables you to choose whether to lock your source design objects while executing this function, and text entry fields which enable you to specify multiple pattern and replacement strings.

Examples

1. This example changes all of the references in all of the versions of the "add_det" component that have the "chref_dir2" pattern in their pathname. After locking the add_det component, this function replaces each occurrence of the chref_dir2 pattern with the chref_dir pattern.

```
$change_design_object_references(
    ["$PROJECT_XYZ/chref_dir/add_det", "mgc_component"],
    ["chref_dir2", "chref_dir"], @lock);
```

2. This example changes all of the references in all of the versions of the “card_reader” component that have the “/idea/user/dpm_rlslib” pattern in their pathname. Without locking the card_reader component, this function replaces each occurrence of the /idea/user/dpm_rlslib pattern with the “\$DPM_DPMRLSLIB” soft prefix.

```
$change_design_object_references(
    ["$PROJECT_ABC/test_dir/card_reader", "mgc_component"],
    ["/idea/user/dpm_rlslib", "$DPM_DPMRLSLIB"], @nolock);
```

3. This example performs the same function as the previous example using command syntax from the popup command line.

```
cha de o r ["$PROJECT_ABC/test_dir/card_reader", "mgc_component"]
    ["/idea/user/dpm_rlslib", "$DPM_DPMRLSLIB"] -n
```

4. This example demonstrates pattern matching for the references of the simple “new_dir” design object. ecause pattern matching stops after a successful match has been found, only a single replacement is made for each reference although each reference matches more than one of the from_pathname_pattern arguments.

```
$change_design_object_references(["$PROJECT_X/new_dir",
    "Mgc_container"], ["A1", "a1", "chref_dir", "CHREF_DIR",
    "DIR", "dir", "B1", "b1", ], @lock);
```

The references of the “new_dir” design object before executing the function:

```
$PROJECT_X/chref_dir/A1
```

```
$PROJECT_X/chref_dir/B1
```

```
$PROJECT_X/DIR/A1
```

```
$PROJECT_X/DIR/B1
```

The references of the “new_dir” design object after executing the function:

```
$PROJECT_X/chref_dir/a1
```

```
$PROJECT_X/CHREF_DIR/B1
```

```
$PROJECT_X/dir/A1
```

```
$PROJECT_X/dir/B1
```

Related Topics

[\\$\\$change_design_object_references\(\)](#)

\$change_location_map_entry()

Scope: session_area

Prerequisite: You can use this function from any Pyxis Project Manager window.

Overrides a location map entry in memory.

Usage

```
$change_location_map_entry("soft_name", "hard_name")
```

CHAnge LOfication Map Entry soft_name hard_name

MGC > Location Map > Change Entry

Arguments

- **soft_name**
A string that specifies the soft prefix whose hard pathname is to be overwritten. This is not a permanent change but only a change to the in-memory location map.
- **hard_name**
A string that specifies the new hard pathname to associate with the soft prefix.

Return Values

VOID.

Description

The interactive function `$change_location_map_entry()` replaces the location map entry which corresponds to the specified soft prefix. The new entry contains only the specified hard pathname. If the location map has not been read into memory, this function forces the map to be read first, before the entry is replaced.

This function replaces the location map entry by matching the soft prefix you specified with a soft prefix in the location map. If a matching soft prefix is found in the location map and its first corresponding hard pathname differs from the `hard_name` argument you specified, this function replaces the hard pathname with the `hard_name` string. If a matching soft prefix is found in the location map and its first corresponding hard pathname is the same as the `hard_name` argument you specified, this function does not perform the replacement.

If you execute the `$change_location_map_entry()` function without specifying the `hard_name` argument, this function displays an error message stating that a hard pathname must be supplied. If a matching soft prefix does not exist in the location map, this function displays an error message stating that a matching soft prefix cannot be found.

Changing the location map using `$change_location_map` only affects the in-memory location map and not the permanent location map. When a session is terminated, changes are lost.

Examples

This example replaces the first hard pathname associated with the *\$DDMS_DOC* soft prefix in the in-memory location map, with the */project/dm/src/docs* hard pathname.

```
$change_location_map_entry("$DDMS_DOC", "/project/dm/src/docs");
```

Location map entry before executing the command:

Soft Name: *\$DDMS_DOC*

Hard Name: *//ddms/local_user/ddms.proj/src/documents.hm*
//project/ddms/src/documents.hm

Location map entry after executing the command:

Soft Name: *\$DDMS_DOC*

Hard Name: */project/dm/src/docs*

Related Topics

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

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

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

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

[\\$get_navigator_directory_hard\(\)](#)

[\\$show_location_map\(\)](#)

\$\$change_object_name()

Scope: dm_config_tk

Renames the specified design object.

Usage

\$\$change_object_name("new_name", "name", "type", *overwrite*)

Arguments

- **new_name**
A string that specifies the new name of the design object.
- **name**
A string that specifies the current name of the design object. You can specify the leaf name or an absolute pathname.
- **type**
A string that specifies the type of the design object. The default is "", which is an empty string.
- **overwrite**
A name that specifies the action taken if a design object with the same name already exists.
Choose one of the following:
 - **@cancel** — Cancels the name change operation. Default.
 - **@replace** — Replaces the conflicting container type design object with the specified design object.

Return Values

VOID.

Description

The toolkit function \$\$change_object_name() renames the specified design object. If you specify the new_name argument as leaf, this function changes the leaf name only. If you specify an absolute pathname, this function moves the design object to the new location. This function changes the references of the specified design object and the references of all contained design objects, to reflect the new leaf name.

If you execute this function on a container type object with the overwrite argument specified as @replace and a target conflict occurs, the conflicting design object is replaced as long as it is also a container type object. The replacement does not occur if either the source or the destination object are not container type objects.

If you execute this function with the overwrite argument specified as `@cancel`, and a target conflict occurs, an error message displays the name and type of the conflicting object and states that the object name could not be changed.

Examples

The following example changes the name of the container “project” to “plan”. If a design object with the name *plan* already exists, this function does not change the name of the design object.

```
$$change_object_name("$PROJECT_XYZ/project",  
    "$PROJECT_XYZ/plan", "Mgc_file", @cancel);
```

Related Topics

[\\$\\$copy_object\(\)](#)

[\\$\\$move_object\(\)](#)

\$change_object_name()

Scope: dmgr_model

Prerequisite: To use this function, you must be in an active Navigator window and must select exactly one design object.

Renames the specified design object.

Usage

\$change_object_name("name", *overwrite*)

CHAnge OBject Name name *overwrite*

Edit > Change > Name

Arguments

- **name**

A string that specifies the new name of the design object. You may specify the name argument as a leaf name, a relative pathname, or an absolute pathname.

- ***overwrite***

A switch that specifies the action taken if a design object with the same name exists. Choose one of the following:

- **@ask_overwrite (-Ask_overwrite)** — Asks for permission to replace the conflicting design object with the selected design object.
- **@cancel (-Cancel)** — Cancels the name change operation.
- **@replace (-Replace)** — Replaces the conflicting container type design object with the selected design object.

Return Values

VOID.

Description

The interactive function \$change_object_name() renames the selected design object. In addition, this function updates all of the design object's contained objects and references to reflect the new name.

If you specify the name argument as a leaf name, this function changes the name of the design object and leaves the design object in the same directory. If you specify a relative pathname, this function changes the name of the design object and places the object in the directory relative to the active navigator directory. If you specify an absolute pathname, the name of the design changes and the object is placed at the location specified by the absolute pathname. If the parent directory in the absolute pathname does not exist, this function returns an error message.

If you use this function to change the name of a component that contains only one symbol with the same name as the component, this function subinvokes the CIB to change the part interface

so that the symbol is automatically renamed to match the component. If there is more than one symbol, or if there is a single symbol but with a name that is different than the component, this function only changes the name of the component.

If you execute this function on a container type object with the overwrite argument specified as `@replace` and a target conflict occurs, the conflicting design object is replaced as long as it is also a container type object. The replacement does not occur if either the source or the destination object are not container type objects.

If you execute this function with the overwrite argument specified as `@cancel`, and a target conflict occurs, an error message displays the name and type of the conflicting object and states that the object name could not be changed.

You can press the `ChangeName` key, as a shortcut method for executing the `$change_object_name()` function, as noted in “[Key Names Mapped to Functions & Scopes](#)” on page 804.

Examples

1. This example changes the name of the selected design object to *build* using the `@replace` value.

```
$change_object_name("build", @replace);
```

2. This example uses a relative pathname and command syntax from the popup command line.

```
cha ob n ../build -replace
```

3. This example uses an absolute pathname and the `@ask_overwrite` value from the popup command line.

```
cha ob n $PROJECT_XYZ/design/joe/build -ask_overwrite
```

Related Topics

[\\$copy_object\(\)](#)

[\\$move_object\(\)](#)

[\\$delete_object\(\)](#)

\$change_object_property()

Scope: dmgr_model

Prerequisite: To use this function, you must be in an active Navigator window and must select exactly one design object.

Changes a property value of the selected design object.

Usage

`$change_object_property("prop_name", "new_value")`

CHAnge OBject Property prop_name new_value

Edit > Change > Property

Arguments

- **prop_name**
A string that specifies the name of the property that this function changes. The string may contain blank spaces.
- **new_value**
A string that specifies the new value of the property.

Return Values

VOID.

Description

The interactive function `$change_object_property()` changes the property value of the selected design object. This function enables you to change the object property of several design objects simultaneously.

If you specify a property that does not exist, this function does not add the property, but it returns a message indicating that the property does not exist. To view a design object's properties, you use the `$report_object_info()` function.

Changing a property of a design object does not create a new version of that design object. To change a property, you must have permission to edit the selected design object. You can use the `$change_protection()` function to change the protection status of design objects.

Examples

1. This example changes the value of the "test_case" property to "A_64".

`$change_object_property("test_case", "A_64");`

2. This example changes the same object property by using command syntax from the popup command line.

cha ob p test_case A_64

Related Topics

[\\$add_object_property\(\)](#)

[\\$delete_object_property\(\)](#)

[Design Object Properties](#)

[\\$report_object_info\(\)](#)

[\\$change_protection\(\)](#)

\$\$change_object_references()

Scope: dm_config_tk

Changes the references of the specified design object.

Usage

```
$$change_object_references([object_list], [pattern_list], version_mode, preview_mode,  
                           lock_mode)
```

Arguments

- **object_list**

A vector of string vectors that specifies the design objects whose references this function changes. The format of this vector is:

```
[["object1", "type1"], ["object2", "type2"], ["objectN", "typeN"]]
```

The object string is the absolute pathname of the design object. The type string specifies the type of the design object.

- **pattern_list**

A vector of string vectors that specify the reference pathname patterns to be replaced. The format of the vector is:

```
["pathname_pattern1", "replace_pattern1", ..., "pathname_patternN",  
 "replace_patternN"]
```

You can specify the `pathname_pattern` string by entering a literal string or by using System V regular expressions.

If the specified pattern occurs more than once in each reference only the first occurrence of the pattern is changed. In the following example, only the first occurrence of "johnj" is replaced with "frankm".

Example reference 1:

```
/usr2/johnj/project_johnj/card_reader
```

Value of `pattern_list` argument:

```
["johnj", "frankm"]
```

Resulting reference:

```
/usr2/frankm/project_johnj/card_reader
```

In the following example, the first and only successful match is the first occurrence of "johnj":

Example reference 2:

```
/usr2/frankm/project_johnj/card_reader
```

Value of pattern_list argument:

[“johnj”, “frankm”, “johnj”, “who”]

Resulting reference:

/usr2/frankm/project_frankm/card_reader

- ***version_mode***

A name that indicates whether the references of all versions or of just the current version are modified by the operation.

Choose one of the following:

- **@current (-Current)** — The references of the current versions of the design objects are modified. Default.
- **@all (-All)** — The references of all versions of the design objects are modified.

- ***preview_mode***

A name that indicates whether the operation actually executes and modifies the object's references, or reports the expected results of an actual execution to the session configuration monitor window, without performing the changes.

Choose one of the following:

- **@nopreview (-Nopreview)** — The operation is performed. Default.
- **@preview (-Preview)** — The results of the operation are reported to the session configuration monitor window, without the operation being performed.

- ***lock_mode***

A name that specifies whether the design object is locked before this function attempts to change its references. This argument can be set to either @nolock or @lock.

Choose one of the following options:

- **@nolock (-Nolock)** — The configuration entry is not locked before this function attempts to change its references.
- **@lock (-Lock)** — The configuration entry is locked before this function attempts to change its references. Default.

Return Values

VOID.

Description

The toolkit function \$\$change_object_references() changes the reference pathnames of either the current version or all versions of all design objects listed in the object_list argument. This function replaces the specified pattern text with the specified replacement text. If the @preview switch is specified, the operation is not actually performed, but the expected results are displayed in the transcript window.

This function changes the references of objects, as they are stored in the reference. Automatic soft-to-hard or hard-to-soft pathname conversion on either the pattern you enter or the stored reference pathname is not performed.

This function and the `$$change_configuration_references()` function can operate on references that are created by other tools. All other Pyxis Project Manager functions can operate on only the design objects with the property “creating_tool”, whose value is “project manager”.

Examples

This example reports the changes that are made when the reference pathname `$PROJECT_XYZ` changes to the new reference pathname `$PROJECT_ABC`, for the current versions of the listed design objects. A regular expression (^) denotes that the pathname must occur at the beginning of the string.

```
$$change_object_references(["$PROJECT_ABC/project",  
    "Mgc_container"], ["^$PROJECT_XYZ", "$PROJECT_ABC"],  
    @current, @nopreview);
```

Related Topics

[`\$\$change_configuration_references\(\)`](#)

\$change_object_references()

Scope: dmgr_model

Prerequisite: To use this function, you must be in an active Navigator window and must select at least one design object.

Changes the references of the selected design object.

Usage

`$change_object_references([pattern_list], version_mode, preview_mode, lock_mode)`

CHAnge OBject References [pattern_list] *version_mode preview_mode lock_mode*

Edit > Change > References

Arguments

- **pattern_list**

A vector of strings that specify the changes to be made in the reference pathname. The format of the vector is:

“pathname_pattern1”, “replace_pattern1”, ...,
“pathname_patternN”, “replace_patternN”]

You can specify the `pathname_pattern` string by entering a literal string or by using System V regular expressions. Note, you cannot use asterisks (wildcard character: “*”).

If the specified pattern occurs more than once in each reference only the first occurrence of the pattern is changed. In the following example, only the first occurrence of “johnj” is replaced with “frankm”:

Example reference 1:

/usr2/johnj/project_johnj/card_reader

Value of `pattern_list` argument:

["johnj", "frankm"]

Resulting reference:

/usr2/frankm/project_johnj/card_reader

In the following example, the first and only successful match is the first occurrence of “johnj”:

Example reference 2:

/usr2/frankm/project_johnj/card_reader

Value of `pattern_list` argument:

["johnj", "frankm", "johnj", "who"]

Resulting reference:

/usr2/frankm/project_frankm/card_reader

- ***version_mode***

A name that specifies whether the references of the current version or of all versions of the object are modified by the operation.

Choose one of the following:

- **@current (-Current)** — The references of the current version of the design object are modified. Default.
- **@all (-All)** — The references of all versions of the design object are modified.

- ***preview_mode***

A name that indicates whether the operation actually executes and modifies the object's references, or only reports the expected results of an actual execution to the session configuration monitor window, without performing the changes.

Choose one of the following:

- **@nopreview (-Nopreview)** — The operation is performed. Default.
- **@preview (-Preview)** — The results of the operation are reported to the session configuration monitor window, without the operation being performed.

- ***lock_mode***

A name that specifies whether the design object is locked before this function attempts to change its references. This argument can be set to either @nolock or @lock.

Choose one of the following options:

- **@nolock (-Nolock)** — The configuration entry is not locked before this function attempts to change its references.
- **@lock (-Lock)** — The configuration entry is locked before this function attempts to change its references. Default.

Return Values

VOID.

Description

The interactive function \$change_object_references() changes the reference pathnames of the specified versions of each selected design object. For each version of the design object, this function edits the reference pathnames, replacing the text as specified in the pattern_list argument. If the @preview switch is specified, the operation is not actually performed, but the expected results of the operation are displayed in the session configuration monitor window.

This function changes the references of objects, as they are stored in the reference. Automatic soft-to-hard or hard-to-soft pathname conversion is not performed on either the pattern you enter or the stored reference pathname.

Changing the references of selected design objects by using the `$change_object_references()` function is an interruptible operation. At any time during the execution of this function, you can press the Kill key to interrupt the operation. When you halt this operation, a dialog box appears prompting you to abort or to continue the operation. If you choose to abort, the operation is halted.

Interrupting the operation, in this way, halts the operation, but the Pyxis Project Manager application does not return the versions of the selected design objects to their original state. Any changes made to the reference pathnames of an object prior to the interruption are permanent, while those reference pathnames that have not been processed prior to the interruption remain unaffected.

This function and the `$change_configuration_references()` function can operate on references that are created by other tools. All other Pyxis Project Manager functions can only operate on design objects with the property “creating_tool”, whose value is “project manager”.

You can press the ChangeReferences key, as a short-cut method for executing the `$change_object_references()` function, as noted in “[Key Names Mapped to Functions & Scopes](#)” on page 804.

Examples

1. This example the reference pathname pattern that matches `$PROJECT_XYZ` changes to `$PROJECT_ABC` for the current version of the selected design object. A regular expression (^) denotes that the pathname must be found at the beginning of the string.

```
$change_object_references(["^$PROJECT_XYZ/project",  
"$PROJECT_ABC/project"], @current, @preview, @lock);
```

2. This example changes removes the dollar sign (\$) from the reference pathnames of all versions of the selected design object by using command syntax from the popup command line.

```
cha ob r ["\$", ""] -all -n -l
```

Related Topics

[\\$add_reference\(\)](#)

[\\$delete_reference\(\)](#)

[\\$change_reference_state\(\)](#)

[\\$report_reference_info\(\)](#)

\$change_password()

Scope: dmgr_proj_tk

Changes the password for administrator mode. This requires write access to the *\$MGC_HOME* tree.

Usage

```
$change_password("oldpassword", "newpassword")
```

Arguments

- **oldpassword**
The current administrator password.
- **newpassword**
The new password. Must be more than one character long.

Return Values

VOID.

Examples

The following example changes the administrator password from the default password to the string "new_password":

```
$change_password("admin1234", "new_password");
```

Related Topics

[\\$login_admin\(\)](#)

[\\$logout_admin\(\)](#)

\$change_protection()

Scope: dmgr_model

Prerequisite: To use this function, you must be in an active Navigator window and must select at least one design object.

Changes the protection of the selected design object.

Usage

\$change_protection("protection")

CHAnge PRotection protection

Edit > Change > Protection

Arguments

- **protection**

A string that specifies the new protection status of the design object.

Return Values

VOID.

Description

The interactive function \$change_protection() changes the protection of the selected design object. Changing a design object's protection enables you to limit access to the design object. When you change the protection of a design object, the protection of each fileset member also changes.

The \$change_protection() function uses UNIX System V protection syntax. UNIX refers to three types of users and three permission settings. The user types are:

- User, or owner of the file (**u**)
- User's group (**g**)
- All others outside of the user's group (**o**)

The permission settings are read (**r**), write (**w**), and execute (**x**).

For example the protection string **o+w** adds write permission to those outside the user's group, using the plus (+) sign. You can remove a permission by using a minus (-) sign.

The protection string **rw-rw-rw** uses the **rw** notation for each type of user. The first **rw** refers to the user, the second refers to the user's group, and the third refers to everyone else. For example, the protection string **rw-r-x---** indicates the user is allowed to read, write, and execute this file; members of the user's group are allowed only to read and execute; and other users are not allowed to read, write, or execute the file.

Examples

1. This example changes the protection of the selected design object to read, write, and execute for the user, the user's group, and all other users.

```
$change_protection("rwxrwxrwx");
```

2. This example changes the protection of the selected design object by using command syntax from the popup command line.

```
cha pr rwxrwxrwx
```

Related Topics

[\\$copy_object\(\)](#)

[\\$move_object\(\)](#)

\$change_reference_property()

Scope: dmgr_reference_model

Prerequisite: To use this function, you must be in an active Reference window and select at least one reference.

Changes a property of the selected reference.

Usage

```
$change_reference_property("prop_name", "prop_value")
```

CHAnge REference Property prop_name prop_value

Arguments

- **prop_name**
A string that specifies the name of the property to be changed. This string may contain blank spaces.
- **prop_value**
A string that specifies the value of the property to be changed.

Return Values

VOID.

Description

The interactive function `$change_reference_property()` changes the specified property of the selected reference. This function enables you to change the reference property of several references simultaneously.

If you specify a property that does not exist, this function does not add the property, but it returns a message indicating that the property does not exist. To view reference properties, you use the `$report_reference_info()` function.

To change a reference property, you must have permission to edit the design object whose references are displayed. You can use the `$change_protection()` function to change the protection status of design objects.

Although this function supports UNIX file system protections, the Pyxis Project Manager application does not provide additional functionality or support for their maintenance. When you change the reference property of a design object, the design object's attribute file receives the owner, group ID and protection of that change process; the other file system objects in the design object maintain their original owner, group ID and protection settings. Changing a reference property does not create a new version of the design object.

Examples

1. This example changes the property of the selected reference.

```
$change_reference_property("Engineer","Steve_Jones");
```

2. This example changes the same property of the selected reference by using command syntax from the popup command line.

```
cha re p Engineer Steve_Jones
```

Related Topics

[\\$add_reference_property\(\)](#)

[\\$select_reference\(\)](#)

[\\$\\$delete_reference_property\(\)](#)

[\\$report_reference_info\(\)](#)

[\\$change_protection\(\)](#)

[Working with Reference Properties](#)

\$change_reference_state()

Scope: dmgr_reference_model

Prerequisite: To use this function, you must be in an active Reference window and must select at least one reference.

Changes the state of the selected reference.

Usage

\$change_reference_state(ref_mode)

CHAnge REference State ref_mode

Arguments

- **ref_mode**

A name that defines the state of the reference.

Choose one of the following:

- **@current** — The reference points to the current version of the target design object. When the version number of the target object is incremented, this reference still points to the current version. Default.
- **@read_only** — The reference points to the current version of the target design object, but the object cannot be edited when accessed through this reference, regardless of the access permissions of the referenced object.
- **@fixed** — Fixed version references are read-only references that point to a particular version of the target design object. As the design object evolves, the fixed reference still points to the version to which it was fixed.

Return Values

VOID.

Description

The interactive function \$change_reference_state() changes the state of the selected reference to current, read-only, or fixed. When you first add a reference using the \$add_reference() function, the reference's state is current.

Changing the state of a reference does not create a new version of the source design object. If you change a reference's state to @fixed, this function points to the current version of the target design object. As the design object evolves, the fixed reference continues to point to the version to which it was originally fixed.

This function only changes references that were added in the Pyxis Project Manager application. To change a reference state, you must have permission to edit the source design object. You can use the \$change_protection() function to change the protection status of design objects.

Examples

1. This example changes the state of the selected reference to fixed.

```
$change_reference_state(@fixed);
```

2. This example changes the state of the selected reference to fixed by using command syntax from the popup command line.

```
cha re s -fixed
```

Related Topics

[`\$add_reference\(\)`](#)

[`\$report_reference_info\(\)`](#)

[`\$change_protection\(\)`](#)

[`\$delete_reference\(\)`](#)

[`\$select_reference\(\)`](#)

[Change References](#)

\$change_version_depth()

Scope: dmgr_model

Prerequisite: To use this function, you must be in an active Navigator window and must select at least one design object.

Changes the version depth of the selected design object.

Usage

\$change_version_depth(depth, *mode*)

CHAnge VERsion Depth depth *mode force*

Edit > Change > Version Depth

Arguments

- **depth**

An integer that specifies the new version depth of the selected design object. The default is the current version depth.

- ***mode***

A switch name that specifies whether the version depth changes should be applied to those design objects that exist within the selected design object's containment or reference hierarchy.

Choose one of the following:

- **@object** — Changes the version depth of the selected design object only. The version depth of the design objects that are either referenced by or contained within the selected design object are not changed. Default.
- **@containment** — Changes the version depth of the selected design object and all of the objects contained within the selected design object.
- **@reference** — Changes the version depth of the selected design object and all of the objects contained within or referenced by either the selected object or by any objects contained within the selected design object.

- ***force (-Force)***

A switch name that suppresses the dialog box. This switch is available only when you execute this function from the command line by using command line syntax.

Return Values

VOID.

Description

The interactive function \$change_version_depth() changes the version depth of the selected design object. When you change the version depth of a design object, this function attempts to

delete previous versions that are beyond the new version depth, but it does not delete previous versions that are frozen. Frozen versions are not included in the total version count.

Specifying the version depth as 0 sets the depth to the default version depth, as specified by the selected design object's type. To avoid completely deleting previous versions, you set the version depth to -1, which enables the design object to have an infinite number of previous versions.

By default, this function changes the version depth of the selected design objects only. You can specify that this function also changes the version depth of all of the objects contained within or referenced by the selected design objects, by specifying the mode argument as either `@containment` or `@reference`.

If you specify the version mode as `@containment`, this function changes the version depth of the selected design objects and of all of the versioned design objects contained within the selected design objects. If you specify the version mode as `@reference`, this function changes the version depth of the selected design objects and of all of the versioned design objects contained within or referenced by the selected design objects or by any object contained within them.

If you select the **Edit > Change > Version Depth** menu item, this function brings up a dialog box that asks for permission to proceed. To suppress the dialog box, you execute this function from the command line by using command syntax.

Changing the version depth of a design object does not create a new version of that design object.

Examples

1. This example changes the selected design object's version depth to 3.

```
$change_version_depth(3, @object);
```

2. This example changes the selected design object's version depth to 3 and suppresses the dialog box, using command syntax.

```
cha ve d 3 -force
```

Related Topics

[\\$\\$delete_version\(\)](#)

[\\$freeze_version\(\)](#)

[Changing Version Depth](#)

\$check_language_views()

Scope: lang_ifc

Finds invalid language or symbol views inside a given project, library, or component.

Usage

`$check_language_views (object_path, object_type)`

Edit > Language > Check Language Views

Arguments

- **object_path**
A string value specifying the project, library, or component's path.
- **object_type**
A string value specifying the project, library, or component's type.

Return Values

Returns a vector of invalid language and/or symbol views paths and an error message, or returns void.

Description

For a given project, library, or component, this function finds invalid language and symbol views inside it. The `$check_language_views()` function runs the `$check_language_view()` command for all registered language views and returns the list of invalid language and symbol views paths.

Examples

The following example checks all the views under the “flow_lib” library and returns a vector of vectors containing invalid views and errors:

```
$check_language_views(["$TESTDIR/test_flow/flow_lib", "mgc_library"])  
  
// Returned value is:  
  
// ["$TESTDIR/test_flow/flow_lib/inv/inv", "mgc_verilog", "Need to be compiled."]]
```

As shown in the returned vector of this example, the Verilog view of the “inv” cell failed the check because it was not compiled after the update.

Related Topics

[\\$show_invalid_language_views\(\)](#)

[\\$check_language_view\(\)](#)

[\\$generate_language_view\(\)](#)

\$check_references()

Scope: dmgr_model

Prerequisite: To use this function, you must be in an active Navigator window and must select at least one design object.

Provides a simplified interface to check and fix the broken references of the selected design objects.

Usage

`$check_references(traverse_refs, broken_limit)`

CHEck REferences *traverse_refs broken_limit*

Edit > Check References

Arguments

- **traverse_refs**

A name that specifies whether references that target objects outside the containment hierarchy are checked. The default is @nottraverse.

Choose one of the following:

- **@nottraverse (-Nottraverse)** — This function does not check the references that target objects outside the containment hierarchy. Default.
- **@traverse (-Traverse)** — This function checks the references that target objects outside the containment hierarchy.

- **broken_limit**

An integer between 1 and 5000 that specifies the maximum number of broken references to be reported. The default is 10.

Return Values

VOID.

Description

The interactive function `$check_references()` checks for broken references in the selected design objects. When broken references are detected, this function displays a dialog box that contains a list of broken references and From and To fields for correcting the broken references. This function only displays broken references that are unique; duplicate broken references are filtered out of the display. You can use this function to interactively check and fix your data's broken references.

The `$check_references()` function uses the configuration toolkit and internally adds all of the selected objects to a configuration and performs a build. The referenced objects of each configuration entry are checked to ensure that they exist. If the target of a reference does not exist, the reference is added to a list of broken references. Reference checking stops when all

references have been checked, or when the number of broken references specified in the `broken_limit` argument have been detected.

When reference checking is complete for either of these reasons and at least one broken reference has been found, the Fix Broken References dialog box displays. The Fix Broken References dialog box contains a list of broken reference pathnames and an expanding list of “From” and “To” input boxes which enable you to change references by entering, for each broken reference, the original pathname and the new replacement pathname. You can fix the broken references by providing the full pathname of the broken reference and the new reference pathname, or by specifying only the portion of the pathname string that you want to be replaced and its replacement pattern.

When you complete your repairs and execute the dialog box, the references are updated and you are presented with the option of verifying whether any broken references remain. If you choose to verify that your changes have fixed the broken references and broken references still exist, the Fix Broken References dialog box is redisplayed. You can repeat this process until all the references are fixed or until you choose not to verify that your changes have fixed the references.

The configuration toolkit only enables a single configuration to be open at one time. When you execute either the `$$open_configuration()` or the `$$create_configuration()` function, any currently open configuration is discarded. Because the `$check_references()` function uses the configuration toolkit to perform its build operation, you need to be sure that there is not an open toolkit configuration when you execute this function, or it is discarded.

You can press the `CheckReferences` key, as a short-cut method for executing the `$check_references()` function, as noted in “[Key Names Mapped to Functions & Scopes](#)” on page 804.

Examples

1. This example checks for up to 15 broken references on the selected design objects and the objects they reference outside their containment hierarchy.

```
$check_references(@traverse, 15);
```

2. This example checks for up to 15 broken references on the selected design objects and the objects they reference outside their containment hierarchy, using command syntax from the popup command line.

```
che re traverse 15
```

Related Topics

[\\$\\$build_configuration\(\)](#)

[\\$set_build_rules\(\)](#)

[\\$\\$change_configuration_references\(\)](#)

\$check_registries()

Scope: dmgr_session_window

Prerequisite: You can use this function from any Pyxis Project Manager window.

Checks the *\$MGC_HOME/registry* directory.

Usage

`$check_registries(use_local_as_shared, "shared_host_name")`

CHEck Registries use_local_as_shared *shared_host_name*

Setup > Check Registries

Arguments

- **use_local_as_shared**

A Boolean that specifies whether to check *\$MGC_HOME/registry* against the local Mentor Graphics Tree directory or against the *\$MGC_HOME* directory specified in the *shared_host_name* argument.

Choose one of the following:

- **@true** — Compares the directories in *\$MGC_HOME/registry* against the local Mentor Graphics Tree's *\$MGC_HOME/shared/registry*. Default.
- **@false** — Compares the directories in *\$MGC_HOME/registry* against the Mentor Graphics Tree specified in the *shared_host_name* argument.

- **shared_host_name**

A string that specifies the name of the workstation whose *\$MGC_HOME/registry* directory you want to compare against. You specify this string with a pathname that is valid for the network and workstation you are using. This string is always dependent on the filesystem conventions that exist for accessing other workstations on your network.

Return Values

VOID.

Description

The interactive function `$check_registries()` invokes the script *\$MGC_HOME/bin/check_rgy*, which compares the directories found in *\$MGC_HOME/registry* with the directories found in *\$MGC_HOME/shared/registry*. This function checks the subdirectories *type_registry*, *font_registry*, *fonts*, and *tcodes*. You use this function to troubleshoot why the Pyxis Project Manager application cannot recognize design objects or icons.

You can check the *\$MGC_HOME/registry* directory against the local Mentor Graphics Tree, or the Mentor Graphics Tree specified in the *shared_host_name* argument.

After this function checks the *\$MGC_HOME/registry* directory, a read-only window displays the results. If the subdirectories are current, the report displays “OK”. If other messages appear, such as “DOES NOT MATCH” or “MISSING”, you should contact your system administrator.

Examples

1. This example checks the directories in *\$MGC_HOME/registry* against the local Mentor Graphics Tree.

```
$check_registries(@true, "");
```

2. This example checks the directories in *\$MGC_HOME/registry* against the Mentor Graphics Tree on the workstation *d2149sa*. This example assumes that the `$check_registries()` function is executed on a Solaris workstation and that */d2149sa* is a valid pathname for accessing the node on the existing network. Specifying a pathname for the `shared_host_name` is always dependent on the network conventions in place.

```
$check_registries(@false, "/d2149sa");
```

\$checkin_objects_to_rc()

Scope: dmgr_project_model

Commits local changes to the repository and performs cancel checkout on objects.

Usage

`$checkin_objects_to_rc([objects], “comment”, recurse)`

Arguments

- **objects**

A vector containing a list of objects to execute with checkin. Each object is itself a vector containing exactly the following two strings:

- The first string is the object pathname.
- The second string is the object type.

- **comment**

A string that specifies the comment to use when adding the project. Default: “No comment.”

- **recurse**

A Boolean that specifies whether checkin occurs hierarchically. Default: @true.

Return Values

VOID.

Description

This function requires that all objects reside in a managed hierarchy. Objects that are unmanaged are added and committed to the repository provided the object does not already exist in the repository.

All managed and modified objects should be checked out by you. Unmodified objects are not checked in; however, cancel checkout is performed on them if they are checked out by you.

Examples

```
$checkin_objects_to_rc([[“$PROJECT/lib/cell”, “mgc_component”]], “Modified cell.”);
```

```
$checkin_objects_to_rc([[“$PROJECT/lib/cell”, “mgc_component”], [“$PROJECT/lib/cat”, \
“mgc_category”]]];
```

Related Topics

[\\$add_project_to_rc\(\)](#)

[\\$create_work_area_from_rc\(\)](#)

[\\$cancel_checkout_objects_for_rc\(\)](#)

[\\$checkout_objects_from_rc\(\)](#)

[\\$update_objects_from_rc\(\)](#)

[\\$object_status_for_rc\(\)](#)

[\\$revert_object_from_rc\(\)](#)

\$checkout_objects_from_rc()

Scope: dmgr_project_model

Checks out objects from the repository to ensure that only one user can make edits on objects.

Usage

`$checkout_objects_from_rc([objects], “comment”, recurse)`

Arguments

- **objects**

A vector containing a list of objects to execute with checkout. Each object is itself a vector containing exactly the following two strings:

- The first string is the object pathname.
- The second string is the object type.

- **comment**

A string that specifies the comment to use when adding the project. Default: “Design Manager checkout.”

- **recurse**

A Boolean that specifies whether checkout occurs hierarchically. Default: @true.

Return Values

VOID.

Description

This function requires that all objects reside in a managed hierarchy. Objects that are unmanaged or already checked out by you are skipped. All managed objects must not be checked out by someone else for this function to succeed.

Examples

```
$checkout_objects_from_rc([[“$PROJECT/lib/cell”, “mgc_component”]], “Need to modified cell.”);
```

```
$checkout_objects_from_rc([[“$PROJECT/lib/cell”, “mgc_component”],  
[“$PROJECT/lib/cat”, “mgc_category”]]);
```

Related Topics

[\\$add_project_to_rc\(\)](#)

[\\$create_work_area_from_rc\(\)](#)

[\\$cancel_checkout_objects_for_rc\(\)](#)

[\\$checkin_objects_to_rc\(\)](#)

[\\$update_objects_from_rc\(\)](#)

[\\$object_status_for_rc\(\)](#)

[\\$revert_object_from_rc\(\)](#)

\$\$clear_entry_filter()

Scope: dm_config_tk

Resets all filters of a primary entry's build rules.

Usage

\$\$clear_entry_filter("name", "type", *version*)

Arguments

- **name**
A string that specifies the pathname of the primary entry.
- **type**
A string that specifies the type of the primary entry.
- **version**
An integer that specifies the version number of the primary entry.

Return Values

VOID.

Description

The toolkit function \$\$clear_entry_filter() resets a primary entry's filters to their default state. After this function executes, the primary entry no longer specifies filters for the inclusion or exclusion of secondary entries. Resetting a primary entry's filters has no effect on its reference traversal setting.

This function resets the following filters:

- **Object path filter** — Filters design objects based on their pathname pattern.
- **Object type filter** — Filters design objects based on their type pattern.
- **Object property filter** — Filters design objects based on object properties.

Examples

1. This example resets all filters for version 2 of the primary entry *schedule*.

```
$$clear_entry_filter("$PROJECT_XYZ/project/schedule",  
    "Document_file", 2);
```

2. To clear only one particular filter, you use the appropriate set filter function without specifications. In this example only the object path filter is reset; other filters are not affected.

```
$$set_object_path_filter([], []);
```

Related Topics

[\\$\\$build_configuration\(\)](#)

[\\$\\$get_object_path_filter\(\)](#)

[\\$\\$get_object_property_filter\(\)](#)

[\\$\\$get_object_type_filter\(\)](#)

[\\$\\$get_reference_property_filter\(\)](#)

[\\$set_build_rules\(\)](#)

[\\$\\$set_object_path_filter\(\)](#)

[\\$\\$set_object_property_filter\(\)](#)

[\\$\\$set_object_type_filter\(\)](#)

[\\$\\$set_reference_property_filter\(\)](#)

\$\$clear_global_status()

Scope: dme_dn_tk

Clears the global toolkit status stack.

Usage

\$\$clear_global_status()

Arguments

None.

Return Values

VOID.

Description

The toolkit function \$\$clear_global_status() clears the global toolkit status stack. The status stack contains the list of status codes that correspond to error messages. Executing this function ensures that no error codes from previous commands exist in the global toolkit status stack.

Examples

The following example clears the global toolkit status stack prior to calling the \$\$move_object() function.

```
function $move_object(destination : string)
{
  $$clear_global_status();
  local object_vector = $get_selected_objects(@full);
  overwrite = @cancel;
  $$move_object(object_vector, destination, overwrite, @nocreate);
  if ($$get_status_code() != 0) {
    $$report_global_status();
    return;
  }
}
```

Related Topics

[\\$\\$get_status_code\(\)](#)

[\\$\\$get_status_messages\(\)](#)

[\\$\\$get_status_code_stack\(\)](#)

[\\$\\$report_global_status\(\)](#)

\$\$clear_monitor()

Scope: dm_config_tk

Prerequisite: To use this function, you must either be in an active Configuration window or in an active configuration monitor window.

Clears all text from the configuration's monitor window.

Usage

\$\$clear_monitor()

Arguments

None.

Return Values

VOID.

Description

The toolkit function \$\$clear_monitor() clears all text from the active configuration's monitor window. If you run this function with some window other than the Configuration window active, the session configuration is the active configuration and its monitor window is cleared. The active configuration's monitor window is automatically cleared upon each new configuration operation.

Although the monitor window is cleared when this function executes, the monitor window does not appear blank until the window is redrawn in one of the following ways:

- By executing the pulldown menu item **MGC > Cleanup Windows**.
- By returning to the Configuration window using the popup menu item **Hide Monitor**, and then reshown the monitor window using the popup menu item **Show Monitor**

You can customize your session setup values to specify that the configuration monitor window is either visible or hidden during configuration operations.

Examples

This example clears all text from the active configuration's monitor window.

```
$$clear_monitor();
```

Related Topics

[\\$hide_monitor\(\)](#)

[\\$\\$writeln_monitor\(\)](#)

[\\$show_monitor\(\)](#)

\$\$close_configuration()

Scope: dm_config_area, dm_config_status_area, dm_config_tk, or dm_config_window

Removes the lock on the active configuration.

Usage

\$\$close_configuration()

Arguments

None.

Return Values

VOID.

Description

The toolkit function \$\$close_configuration() removes the lock on the active configuration. After this function executes, there is no active configuration until you call the \$\$open_configuration() function or the \$\$create_configuration() function.

This function does not save changes made to the configuration. You must first use the \$\$save_configuration() function or the \$\$save_configuration_as() function to save any changes before you close the configuration.

Examples

The following example opens the configuration *config_alpha*, adds the entry *base*, and then saves and closes the configuration.

```
$$open_configuration("$PROJECT_XYZ/config_alpha", 0, @read_write);  
  
$$add_configuration_entry("$PROJECT_XYZ/example/base", "Mgc_file", 2);  
  
$$save_configuration();  
  
$$close_configuration();
```

Related Topics

[\\$\\$build_configuration\(\)](#)

[\\$\\$save_configuration\(\)](#)

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

[\\$\\$save_configuration_as\(\)](#)

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

\$close_hierarchy()

Scope: dmgr_project_model

Closes the hierarchy at the specified path if it is currently open in the Project Navigator window.

Usage

```
$close_hierarchy("path");
```

CLOse HIerarchy

File > Close Hierarchy

Arguments

- **path**

The path of the hierarchy to be closed.

This must be either the soft path to the hierarchy or a hard path that can be resolved using the hierarchy's location map.

Return Values

VOID.

Examples

The following example closes a project located at */tmp/project*:

```
$close_hierarchy("/tmp/project");
```

Related Topics

[\\$\\$open_hierarchy\(\)](#)

[\\$maintain_hierarchy\(\)](#)

\$\$close_versioned_object()

Scope: dme_do_tk

Closes a versioned design object that encapsulates data from a non-Mentor Graphics application.

Usage

```
$$close_versioned_object("obj_name", "obj_type", save)
```

Arguments

- **obj_name**
A string that specifies the pathname of the versioned design object to be closed.
- **obj_type**
A string that specifies the type of the versioned design object to be closed.
- **save**
A name that specifies the action taken when a save is performed.
Choose one of the following:
 - **@save** — Closes and saves any changes of the versioned design object. Default.
 - **@forget** — Closes and restores the versioned design object to its original state.

Return Values

A Boolean that indicates if the versioned design object was closed. If the design object was successfully closed, this function returns @true; otherwise, this function returns @false.

Description

The toolkit function \$\$close_versioned_object() closes a versioned design object and renames the design object in conformance with the Mentor Graphics versioning scheme. You should use this function in conjunction with the \$\$create_versioned_object() and \$\$open_versioned_object() functions. This set of functions enable you to apply the Mentor Graphics version management scheme to design objects that encapsulate data from non-Mentor Graphics applications.

Note



Note that the use of the \$\$close_versioned_object(), \$\$create_versioned_object(), and \$\$open_versioned_object() functions is not always required for encapsulating data from non-Mentor Graphics applications. The only situation where these functions are required is for managing *versions* of data from non-Mentor Graphics applications. For information about encapsulating data, refer to the *Pyxis Registrar User's and Reference Manual* on SupportNet.

When this function closes the design object, it does the following:

- Creates or updates the attribute file
- Saves or forgets any edits
- Increments the version number
- Removes the lock from the design object

If you used the `$$create_versioned_object()` function, then this function creates the attribute file. If you used the `$$open_versioned_object()` function, then this function updates the existing attribute file.

By default, this function saves any changes made to the design object. To forget changes, you must specify the `@forget` value. If you call the `$$create_versioned_object()` function and specify the save argument as `@forget`, any changes made to the encapsulated tool's data files are left unmodified. If you call the `$$open_versioned_object()` function and specify the save argument as `@forget`, then the encapsulated tool's data files are deleted.

Examples

Assume that you have a third-party tool that works on files with the *.hck* suffix, and that the *Phantom* type has the *.hck* fileset member. You have registered the *Phantom* type and the third-party tool in your local registry.

The following example closes the “my_file” design object of “Phantom” type.

```
$$close_versioned_object("$PROJECT_XYZ/my_file", "Phantom");
```

After closing the design object, *my_file.hck* is renamed *my_file.hck_1*. At this point, the following files exist:

```
my_file.Pantom.attr  
my_file.hck_1
```

If you later open the design object by using `$$open_versioned_object()`, make changes to the data, and then close the object by using `$$close_versioned_object()`, the following files exist:

```
my_file.Pantom.attr  
my_file.hck_1  
my_file.hck_2
```

Related Topics

[\\$\\$create_versioned_object\(\)](#)

[\\$\\$open_versioned_object\(\)](#)

\$close_window()

Scope: dm_config_status_area, dm_config_window, dm_iconic_area, dm_list_area, dmgr_session_window, dmgr_tool_area, dmgr_toolbox_area, dmgr_type_area

Closes the active window.

Usage

`$close_window(action)`

CLOse WInow action

Window Menu: Close

Arguments

- *action*

A switch that specifies whether changes made in the window are saved or discarded.

Choose one of the following:

- **@ask (-Ask)** — Prompts you to specify whether you want to save or discard your edits before exiting the window. Default.
- **@save (-Save)** — Saves the edits in the window that is being closed.
- **@discard (-Discard)** — Discards the unsaved edits in the window that is being closed.

Return Values

A Boolean that indicates if the window was closed. If the window was successfully closed, this function returns @true; otherwise, this function returns @false.

Examples

1. This example closes the active window and discards any unsaved edits.

```
$close_window(@discard);
```

2. This example closes the active window and discards any unsaved edits by using command syntax from the popup command line.

```
clo wi -discard
```

Related Topics

[\\$open_configuration_window\(\)](#)

[\\$open_tools_window\(\)](#)

[\\$open_navigator\(\)](#)

[\\$open_trash_window\(\)](#)

\$compile_all_models()

Scope: lang_ifc

Compiles the currently selected project, library, cell, or HDL source file.

Usage

`$compile_all_models()`

Edit > Language > Compile

Arguments

None.

Return Values

VOID.

Examples

The following example compiles the currently selected HDL isource file:

```
$compile_all_models();
```

\$\$convert_configuration_references()

Scope: dm_config_tk

Converts the references of each entry in the active configuration.

Usage

`$$convert_configuration_references(preview_mode, lock_mode)`

Arguments

- ***preview_mode***

A name that indicates whether the operation actually executes and modifies the configuration's references, or reports the expected results of an actual execution to the session configuration monitor window, without performing the changes.

Choose one of the following:

- **@nopreview** — The operation is performed. Default.
- **@preview** — The results of the operation are reported to the session configuration monitor window, without the operation being performed.

- ***lock_mode***

A name that specifies whether the configuration object is locked before this function attempts to change its references.

Choose one of the following options:

- **@nolock** — The configuration entry is not locked before this function attempts to change its references.
- **@lock** — The configuration entry is locked before this function attempts to change its references. Default.

Return Values

VOID.

Description

The toolkit function `$$convert_configuration_references()` changes the references on the open configuration object from hard pathnames to soft pathnames. The soft pathnames used are based on the current location map. If the `@preview` switch is specified, the operation is not actually performed, but the expected results are displayed in the transcript window.

This function and the `$$convert_object_references()` function can operate on references that are created by other tools. All other Pyxis Project Manager functions can operate on only the design objects with the property “creating_tool”, whose value is “project manager”.

Examples

This example changes all references on the active configuration to soft pathnames.

```
$$convert_configuration_references(@nopriview, @lock);
```

Related Topics

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

\$convert_configuration_references()

Scope: dm_config_window

Automatically converts reference pathnames on all objects in the configuration from hard reference pathnames to soft reference pathnames using the active location map.

Usage

`$convert_configuration_references(preview_mode, lock_mode, report_warnings)`

CONvert CONfiguration References *preview_mode lock_mode report_warnings*

Edit > Convert References

Arguments

- ***preview_mode***

A name that indicates whether the operation actually executes and modifies the configuration's references, or reports the expected results of an actual execution to the session configuration monitor window, without performing the changes.

Choose one of the following:

- **@nopreview (-Nopreview)** — The operation is performed. Default.
- **@preview (-Preview)** — The results of the operation are reported to the session configuration monitor window, without the operation being performed.

- ***lock_mode***

A name that specifies whether the configuration object is locked before this function attempts to change its references. This argument can be set to either @nolock or @lock.

Choose one of the following options:

- **@nolock (-NOLock)** — The configuration entry is not locked before this function attempts to change its references.
- **@lock (-Lock)** — The configuration entry is locked before this function attempts to change its references. Default.

- ***report_warnings***

A name that specifies whether unconverted references are reported to the monitor window.

- **@nowarnings (-NOWarnings)** — No information about the conversion is reported. Default.
- **@warnings (-Warnings)** — References whose hard pathnames were not successfully converted to soft pathnames are reported.

Return Values

VOID.

Description

The toolkit function `$convert_configuration_references()` changes the references on the open configuration object from hard pathnames to soft pathnames. Because the values in the current location map are used to perform the conversion, this function does not run if a location map is not loaded. If you attempt to execute this function and a location map is not loaded, a message displays stating that you must use the `$read_map()` function to load a location map before you can execute this function. If the `@preview` switch is specified, the operation is not actually performed, but the expected results are displayed in the transcript window.

This function and the `$convert_object_references()` function can operate on references that are created by other tools. All other Pyxis Project Manager functions can operate on only the design objects with the property “creating_tool”, whose value is “project manager”.

Examples

1. This example changes all references on the active configuration to soft pathnames and reports all unconverted references.

```
$convert_configuration_references(@nopriview,@lock,@warnings);
```

2. This example changes all references on the active configuration to soft pathnames using command syntax and does not report unconverted references to the monitor window.

```
con co r -n -l -now
```

Related Topics

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

\$\$convert_object_references()

Scope: dm_config_tk

Converts the references of the specified design object(s).

Usage

`$$convert_object_references([object_list], version_mode, preview_mode, lock_mode)`

Arguments

- **object_list**

A vector of string vectors that specifies the design objects whose references this function changes. The format of this vector is:

`[["object1", "type1"], ["object2", "type2"], ["objectN", "typeN"]]`

The object string is the absolute pathname of the design object. The type string specifies the type of the design object.

- **version_mode**

A name that indicates whether the references of all versions or of just the current version are modified by the operation.

Choose one of the following:

- **@current** — The references of the current versions of the design objects are modified. Default.
- **@all** — The references of all versions of the design objects are modified.

- **preview_mode**

A name that indicates whether the operation actually executes and modifies the object's references, or reports the expected results of an actual execution to the session configuration monitor window, without performing the changes.

Choose one of the following:

- **@nopreview** — The operation is performed. Default.
- **@preview** — The results of the operation are reported to the session configuration monitor window, without the operation being performed.

- **lock_mode**

A name that specifies whether the design object is locked before this function attempts to change its references. This argument can be set to either **@nolock** or **@lock**.

Choose one of the following options:

- **@nolock** — The design object is not locked before this function attempts to change its references.

- **@lock** — The design object is locked before this function attempts to change its references. Default.

Return Values

VOID.

Description

The toolkit function `$$convert_object_references()` changes the reference pathnames of specified design objects from hard pathnames to soft pathnames. The values in the current location map are used to perform the conversion. If the `@preview` switch is specified, the operation is not actually performed, but the expected results are displayed in the transcript window.

This function and the `$$convert_configuration_references()` function can operate on references that are created by other tools. All other Pyxis Project Manager functions can operate on only the design objects with the property *“creating_tool”*, whose value is *“project manager”*.

Examples

This example changes the references on the *\$PROJECT_ABC/project* design object to soft pathnames and reports all non-converted references.

```
$$convert_object_references([["$PROJECT_ABC/project",  
    "Mgc_container"]], @current, @nopriview, @lock, @warnings);
```

Related Topics

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

\$convert_object_references()

Scope: dmgr_model

Prerequisite: A design object must be selected before this function can be executed.

Converts the reference pathnames of selected design objects automatically from hard to soft using the current location map.

Usage

`$convert_object_references(version_mode, preview_mode, lock_mode, report_warnings)`

CONvert OBJect References *version_mode preview_mode lock_mode report_warnings*

Edit > Convert Refs

Navigator > Edit > Convert Refs

Arguments

- ***version_mode***

A name that indicates whether the references of all versions or of just the current version are modified by the operation.

Choose one of the following:

- **@current (-Current)** — The references of the current versions of the design objects are modified. Default.
- **@all (-All)** — The references of all versions of the design objects are modified.

- ***preview_mode***

A name that indicates whether the operation actually executes and modifies the object's references, or reports the expected results of an actual execution to the session configuration monitor window, without performing the changes.

Choose one of the following:

- **@nopreview (-Nopreview)** — The operation is performed. Default.
- **@preview (-Preview)** — The results of the operation are reported to the session configuration monitor window, without the operation being performed.

- ***lock_mode***

A name that specifies whether the design object is locked before this function attempts to change its references. This argument can be set to either @nolock or @lock.

Choose one of the following options:

- **@nolock (-NOLock)** — The design object is not locked before this function attempts to change its references.
- **@lock (-Lock)** — The design object is locked before this function attempts to change its references. Default.

- ***report_warnings***

A name that specifies whether unconverted references are reported to the monitor window.

- **@nowarnings (-NOWarnings)** — No information about the conversion is reported. Default.
- **@warnings (-Warnings)** — References whose hard pathnames were not successfully converted to soft pathnames are reported.

Return Values

VOID.

Description

The function `$convert_object_references()` changes the reference pathnames of specified design objects from hard pathnames to soft pathnames. Because the values in the current location map are used to perform the conversion, this function does not run if a location map is not loaded. If you attempt to execute this function and a location map is not loaded, a message displays stating that you must use the `$read_map()` function to load a location map before you can execute this function. If the `@preview` switch is specified, the operation is not actually performed, but the expected results are displayed in the transcript window.

This function and the `$convert_configuration_references()` function can operate on references that are created by other tools. All other Pyxis Project Manager functions can operate on only the design objects with the property “creating_tool”, whose value is “project manager”.

Examples

1. This example changes the references on the selected design object(s) to soft pathnames and reports all unconverted references to the monitor window.

```
$convert_object_references(@current,@nopreview,@lock,@warnings);
```

2. This example changes the references on the selected design object(s) to soft pathnames using command syntax and does not report unconverted references to the monitor window.

```
con ob r -c -n -l -now
```

Related Topics

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

\$\$copy_configuration()

Scope: dm_config_tk

Copies all design object versions in the active configuration, to a target location.

Usage

\$\$copy_configuration("target", *overwrite*, *create_dest*)

Arguments

- **target**

A string that specifies the pathname of the destination directory. You can specify a relative or an absolute pathname. If you specify a relative pathname, the Pyxis Project Manager application assumes that the path is relative to the current working directory.

- **overwrite**

A name that specifies the action that is taken if conflicting design objects exist at the destination directory.

Choose one of the following:

- **@cancel** — Cancels the copy operation. Default.
- **@replace** — Replaces the conflicting container type design objects.

- **create_dest**

A name that specifies the action taken if the destination directory does not exist.

Choose one of the following:

- **@ncreate** — Does not create the destination directory and cancels the copy operation. Default.
- **@create** — Creates the destination directory and proceeds with the copy operation.

- **stop_on_err**

A name that specifies the action taken if an error is encountered during the copy.

Choose one of the following:

- **@no_stop** — Does not cancel the copy operation. Default.
- **@stop** — Cancels the copy operation.

- **update_refs**

A name that specifies the action taken for updating the references to the copied object.

Choose one of the following:

- **@update** — Updates all references to the copied configuration. Default.
- **@noupdate** — Does not update references for the configuration.

Return Values

VOID.

Description

The toolkit function `$$copy_configuration()` copies the design object versions in the active configuration to a target location.

By default, this function cancels the copy operation if conflicting design objects exist in the destination directory. To replace conflicting design objects, you must specify `@replace` as the overwrite argument.

If you execute this function on a container type object with the overwrite argument specified as `@replace` and a target conflict occurs, the conflicting design object is replaced as long as it is also a container type object. The replacement does not occur if either the source or the destination object are not container type objects.

If the destination directory does not exist, this function cancels the copy operation and returns a message indicating that the directory you specified does not exist. To create the destination directory automatically, you must specify `@create` as the `create_dest` argument.

If you set the target path for a retargetable entry, that entry is copied to the location specified in the target path and not to the destination directory specified for the copy operation.

Examples

The following example opens the configuration *config_a* and copies it to the target location *\$PROJECT_XYZ/design2/project*. If conflicting design objects exist in the target location, they are replaced. If the target location does not exist, it is created.

```
$$open_configuration("$PROJECT_XYZ/design/config_a", 0,  
    @read_write);  
  
$$copy_configuration("$PROJECT_XYZ/design2/project",  
    @replace, @create);
```

Related Topics

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

[\\$\\$set_target_path\(\)](#)

\$copy_configuration()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window.

Copies all design object versions in the active Configuration window to a target location.

Usage

`$copy_configuration("destination_directory", overwrite_mode, preview)`

COPY COnfiguration destination_directory *overwrite_mode* *preview*

Edit > Copy Configuration

Configuration window popup menu > **Global Operations > Copy**

Configuration window toolbar > **Copy Configuration**

Arguments

- **destination_directory**

A string that specifies the absolute pathname of the target location.

- **overwrite_mode**

A switch name that specifies the action taken if conflicting design objects exist in the destination directory.

Choose one of the following:

- **@cancel (-Cancel)** — Cancels the copy operation. Default.
- **@replace (-Replace)** — Replaces the conflicting container type design objects.

- **preview**

A switch name that specifies if this function issues a preview report.

Choose one of the following:

- **@nopreview (-Nopreview)** — Copies the configuration but does not issue the preview report. Default.
- **@preview (-Preview)** — Issues the preview report but does not copy the configuration.

- **stop_on_err**

A name that specifies the action taken if an error is encountered during the copy.

Choose one of the following:

- **@no_stop** — Does not cancel the copy operation. Default.
- **@stop** — Cancels the copy operation.

- ***update_refs***

A name that specifies the action taken for updating the references to the copied object.

Choose one of the following:

- **@update** — Updates all references to the copied configuration. Default.
- **@noupdate** — Does not update references for the configuration.

Return Values

VOID.

Description

The interactive function `$copy_configuration()` copies the design object versions in the active Configuration window to the directory specified in the `destination_directory` argument. After the copy is complete, this function updates references of the design objects and any objects they contain, to reflect their new location.

If the destination directory that you specify does not exist, this function asks for permission to create the new directory. By default, this function does not copy the configuration if conflicting design objects exist in the destination directory.

To replace the conflicting design objects, you specify `@replace` as the `overwrite_mode` argument. If you execute this function on a container type object with the `overwrite` argument specified as `@replace` and a target conflict occurs, the conflicting design object is replaced as long as it is also a container type object. The replacement does not occur if either the source or the destination object are not container type objects.

The preview mode displays a read-only window that contains the absolute pathname of the design object version to be copied, and the absolute pathname of the new location, including any conflicts that might occur. To preview the copy, you specify `@preview` as the `preview` argument.

Copying a configuration by using the `$copy_configuration()` function is an interruptible operation. At any time during the execution of this function, you can press the Kill key to interrupt the operation. When you halt the copy operation, a dialog box appears prompting you to abort or to continue the operation. If you choose to abort, the operation is halted.

Interrupting the copy operation in this way halts the operation, and the Pyxis Project Manager application attempts to return the configuration to its original state by deleting the directories created by the function. In some cases, particularly if the interrupt is executed late in the operation, the clean-up procedures might not successfully remove all the directories created by this function at the source or destination; however, the source configuration object always remains undamaged.

While executing, the `$copy_configuration()` function writes a status summary to the configuration's monitor window. The summary includes the number of failures, errors, and warnings found during the operation. Additionally, a message appears in the message area, stating whether the copy operation failed, passed, or passed with warnings.

If your session setup values specify to show the configuration monitor window, the monitor window is opened when you execute this function. When the operation is complete, the monitor window remains visible until you return to the Configuration window by executing the monitor window's popup menu item **Hide Monitor**. You can redisplay the operation's status summary by executing the Configuration window's popup menu item **Show Monitor**.

The configuration's monitor window is cleared upon the execution of a new configuration operation. To save the status information in a file, you can use the Configuration window popup menu item **Export**.

If you set the target path for a retargetable entry, that entry is copied to the location specified in the target path and not to the destination directory specified for the copy operation.

Examples

1. This example successfully copies the contents of the active Configuration window to the destination directory *\$PROJECT_XYZ/config2*.

```
$copy_configuration("$PROJECT_XYZ/config2",@cancel,@nopreview);
```

The following status information is reported to the monitor window:

```
Copying Configuration...
Checking for Conflicts...
  Conflict Resolution Complete
  Processing Entries...
  Entry Processing Complete
  Updating References...
  Reference Update Complete
Copy Complete
```

2. This example copies the contents of the active Configuration window by using command syntax from the popup command line.

```
cop co $PROJECT_XYZ/config2 -cancel -nopreview
```

Related Topics

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

[\\$set_target_path\(\)](#)

[Change Your Session Setup](#)

\$\$copy_design_object()

Scope: dme_base_tk

Copies one or more design objects to a container.

Usage

`$$copy_design_object([from], "to_string", overwrite, create_dest, change_references)`

Arguments

- **from**

A vector of string vectors which specifies the design objects that are to be copied. The format of this vector of string vectors is:

`[["object1", "type1"], ["object2", "type2"], ..., ["objectN", "typeN"]]`

The object is a string that specifies the absolute pathname of the design object. The type is a string that specifies the type of the design object.

- **to_string**

A string that specifies the destination container for the design objects that are copied. The string can be a relative or an absolute pathname. If a relative pathname is specified, it is converted to an absolute pathname using the value of the MGC_WD environment variable before being stored in the reference.

- **overwrite**

A name that specifies the action taken if conflicting design objects exist in the destination container. This argument can be set to either `@cancel` or `@replace`. The default is either the last option specified or, if you have not executed this function in the current session, `@cancel`. The `$$copy_design_object()` function *never* overwrites a container.

Choose one of the following options:

- **@cancel (-Cancel)** — If a design object exists in the destination container that has the same name and type as the design object being copied, the copy operation is canceled and none of the specified design objects are copied to the destination container. An error message reports a target conflict and displays the name of the conflicting design object.
- **@replace (-Replace)** — When a design object exists in the destination container with the same name and type as a design object being copied, the conflicting design object in the destination container is replaced with the source design object, if the conflicting object is not a container design object.

- **create_dest**

A name that specifies the action taken when the specified destination container does not exist prior to the execution of the `$$copy_design_object()` function. This argument can be set to either `@ncreate` or `@create`.

Choose one of the following options:

- **@ncreate (-NOCReate)** — If the destination container does not exist when this function executes, the copy operation is not performed and an error message reports that the specified destination container does not exist. Default.
- **@create (-CReate)** — If the destination container does not exist when this function executes, the `$$copy_design_object()` function attempts to create the container and to complete the copy operation. If the attempt is unsuccessful, none of the specified source design objects are copied and an error message reports the reason that the destination container could not be created.

- ***change_references***

A name that specifies whether this function should attempt to update references or leave the references exactly as they were in the source design object.

Choose one of the following options:

- **@nochange (-NOCHange)** — None of the references of the design objects being copied are changed by the copy operation. When the copy operation is complete, all of the references of the copied design objects still points to the target object they referenced before the copy operation.
- **@change (-CHange)** — Any references in the design objects being copied that point to design objects that are also being copied, are changed to point to the new pathname of their target design object. Default.

Return Values

An integer that indicates if the specified design objects were successfully copied. If the design objects were successfully copied, this function returns a 0 for successful completion; otherwise, this function returns a 1 for error.

Description

The iDM toolkit function `$$copy_design_object()` copies all of the versions of each of the specified design objects to the specified destination container. This function automatically updates the references between copied design objects, unless you specify the `change_references` option as `@nochange`.

If you specify the `to_string` as a relative pathname, the relative pathname is converted to an absolute pathname using the value of the `MGC_WD` environment variable. If this environment variable is not set, the current container at the time your application was invoked is used to resolve the relative pathname. If you have reset the current container using the `$set_working_directory()` function, the container that you specified with this function is used to resolve the relative pathname.

If you specify a single source design object to copy and you specify the destination using a new leaf name, this function copies the source design object to the destination container and renames the design object with the newly specified leaf name.

By default, this function cancels the copy operation if the specified destination container does not exist. You can specify that a destination container should be created when you copy multiple design objects to a destination container that does not exist by specifying the `create_dest` argument as `@create`. However, if you specify `@create` when copying a single design object, the `@create` option does not create a new container but, instead, renames the design object to the non-existent leaf name. If any segment of your pathname except for the leaf name does not exist, the `@create` option has no effect and the copy operation fails.

By default, the `$$copy_design_object()` function cancels the copy operation if conflicting design objects exist in the destination container. To replace the conflicting design objects, you specify the `overwrite` argument as `@replace`. When you specify `@replace`, this function overwrites any conflicting design objects that are not either containers or directories.

Container type design objects are never overwritten. If you execute this function using the `@replace` option and a conflicting container design object exists in the target destination, this function copies all of the design objects specified in the `from` argument except for the conflicting *container* design object and displays a message stating that the container design object was not overwritten.

Examples

1. This example copies the *add_det* symbol to a test directory without changing its references to point to its new location. If an *add_detector* container does not exist in the test location, one is created with the `@create` option. If an *add_det* symbol already exists in the *add_detector* container, the copy operation is canceled.

```
$$copy_design_object(["$PROJECT_XYZ/chref_dir/add_det/add_det",  
"mgc_symbol"], "/user/phc/test/add_detector", @cancel,  
@create, @nochange);
```

2. This example copies the *analog* component to a new project directory and updates its references to the new location. If the *card_reader* container does not exist, the copy operation is canceled and the *analog* component is not copied. If a design object with the name "analog" exists in the destination directory, the copy operation is canceled and the *analog* component is not copied.

```
$$copy_design_object(["$PROJECT_XYZ/dpm/card_reader/analog",  
"mgc_component"], "$PROJECT_ABC/chref_dir/card_reader",  
@replace, @ncreate, @change);
```

Related Topics

[\\$copy_design_object\(\)](#)

\$copy_design_object()

Scope: dme_base_tk

Prerequisite: To use this function, you must be in an active Navigator window.

Copies one or more design objects to a container.

Usage

`$copy_design_object([from], "to_string", overwrite, create_dest, change_references)`

COPY DDesign Object [from] to_string *overwrite create_dest change_references*

Edit > Copy To

Arguments

- **from**

A vector of string vectors which specifies the design objects that are to be copied. The format of this vector of string vectors is:

`[["object1", "type1"], ["object2", "type2"], ..., ["objectN", "typeN"]]`

The object is a string that specifies the absolute pathname of the design object. The type is a string that specifies the type of the design object.

- **to_string**

A string that specifies the destination container for the design objects that are copied. The string can be a relative or an absolute pathname. If a relative pathname is specified, it is converted to an absolute pathname using the value of the MGC_WD environment variable before being stored in the reference.

- **overwrite**

A name that specifies the action taken if conflicting design objects exist in the destination container. This argument can be set to either `@cancel` or `@replace`. The default is the last option specified or, if you have not executed this function in the current session, `@cancel`. The `$copy_design_object()` function *never* overwrites a container.

- **@cancel (-Cancel)** — If a design object exists in the destination container that has the same name as the design object being copied, the copy operation is canceled and none of the specified design objects are copied to the destination container. An error message reports a target conflict and displays the name of the conflicting design object.
- **@replace (-Replace)** — When a design object exists in the destination container with the same name as a design object being copied, the conflicting design object in the destination container is replaced with the source design object if it is not a container design object. To ensure the integrity of your data, container design objects are never overwritten.

- ***create_dest***

A name that specifies the action taken when the specified destination container does not exist prior to the execution of the \$copy_design_object() function. This argument can be set to either @ncreate or @create.

Choose one of the following options:

- **@ncreate (-NOCReate)** — If the destination container does not exist when this function executes, the copy operation is not performed and an error message reports that the specified destination container does not exist. Default.
- **@create (-CReate)** — If the destination container does not exist when this function executes, the \$copy_design_object() function attempts to create the container and to complete the copy operation. If the attempt is unsuccessful, none of the specified source design objects are copied and an error message reports the reason that the destination container could not be created.

- ***change_references***

A name that specifies whether this function should attempt to update references or leave the references exactly as they were in the source design object.

Choose one of the following options:

- **@nochange (-NOCHange)** — None of the references of the design objects being copied are changed by the copy operation. When the copy operation is complete, all of the references of the copied design objects still points to the target object they referenced before the copy operation.
- **@change (-CHange)** — Any references in the design objects being copied that point to design objects that are also being copied, are changed to point to the new pathname of their target design object. Default.

Return Values

VOID.

Description

The iDM interactive function \$copy_design_object() copies all of the versions of each of the specified design objects to the specified destination container. This function automatically updates the references between copied design objects, unless you specify the change_references option as @nochange. If you specify the to_string argument as a relative pathname, the relative pathname is converted to an absolute pathname using the value of the MGC_WD environment variable. If this environment variable is not set, the current directory at the time your application was invoked is used to resolve the relative pathname. If you have reset the current directory using the \$set_working_directory() function, the directory that you specified with this function is used to resolve the relative pathname.

If you specify a single source design object to copy and you specify the destination container using a new leaf name, this function copies the source design object to the destination container and renames the design object with the newly specified leaf name.

By default, this function cancels the copy operation if the specified destination container does not exist. You can specify that a destination container should be created when you copy multiple design objects to a destination container that does not exist by specifying the `create_dest` argument as `@create`. However, if you specify `@create` when copying a single design object, the `@create` option does not create a new container but, instead, renames the design object to the non-existent leaf name. If any segment of your pathname except for the leaf name does not exist, the `@create` option has no effect and the copy operation fails.

By default, the `$copy_design_object()` function cancels the copy operation if conflicting design objects exist in the destination container. To replace the conflicting design objects, you specify the `overwrite` argument as `@replace`. When you specify `@replace`, this function overwrites any conflicting design objects that are not either containers or directories.

Container type design objects are never overwritten. If you execute this function using the `@replace` option and a conflicting container design object exists in the target destination, this function copies all of the design objects specified in the `from` argument except for the conflicting *container* design object and displays a message stating that the container design object was not overwritten.

When you execute the `$copy_design_object_references()` function using either the menu interface or command syntax, the Copy Design Object References dialog box displays enabling you to easily enter your argument selections.

The Copy Design Object References dialog box contains two dialog navigators that enable you to select the design objects you want to copy and to specify the destination container to which you want to copy them. The dialog box also provides a filter option for each of the dialog navigators which enables you to limit the set of design objects that you see displayed in each of the dialog navigators and an **Options** button which provides you with options for specifying your overwrite, destination creation, and reference update options.

You can interrupt the execution of the `$copy_design_object()` function at any time by pressing the Kill key. When you interrupt this function, you are prompted if you want to abort the current operation. When you choose to continue with the abort operation, you are returned to the session from which you executed this function.

Examples

1. This example copies the part interface and the schematic model for the “add_det” component to the “card_reader” component and updates their references to reflect their new location. If the `card_reader` container does not exist, it is not created and the copy operation is canceled. If a design object with the name `part` or `schematic` exists in the `card_reader` container prior to the copy operation, the copy operation is canceled.

```
$copy_design_object(["$PROJECT_XYZ/chref_dir/add_det/part",  
"Eddm_part"],["$PROJECT_XYZ/chref_dir/add_det/schematic",  
"mgc_schematic"], "$PROJECT_XYZ/dpm/card_reader", @cancel,  
@nocreate, @change);
```

2. This example performs the same function as the previous example using command syntax from the popup command line.

```
cop de o ["$PROJECT_XYZ/chref_dir/add_det/part", "Eddm_part"]  
["$PROJECT_XYZ/chref_dir/add_det/schematic", "mgc_schematic"]  
$PROJECT_XYZ/dpm/card_reader -c -nocr -ch
```

Related Topics

[\\$\\$copy_design_object\(\)](#)

\$\$copy_object()

Scope: dm_config_tk

Copies the specified design object to a target location.

Usage

`$$copy_object([object_list], "target", [filter_list], follow_refs, preview_mode, copy_mode, overwrite, create_dest, update)`

Arguments

- **object_list**

A vector of string vectors that specifies the design objects to be copied. The format of the vector of string vectors is:

```
[["name1", "type1", version], ["name2", "type2", version], ..., ["nameN", "typeN", version]]
```

The name is a string that specifies the absolute pathname of the design object. The type is a string that specifies the type of the design object. Version is an optional integer.

- **target**

A string that specifies the pathname of the destination directory. You can specify a relative or an absolute pathname. If you specify a relative pathname, the Pyxis Project Manager application assumes the path is relative to the current working directory.

- **filter_list**

A vector of string vectors which specifies which objects are to be excluded from the release, based on their pathname.

```
["string1", "string2", "stringN"]
```

You can specify the string pattern by entering a literal string or by using UNIX System V regular expressions. The default is to include all objects in the release.

- **follow_refs**

A name that enables you to set reference traversal to either on or off. If the selected object is a container design object, all objects contained within the container are copied and their references are updated. Objects that are referenced by the selected object or by any object contained within its containment may be included or excluded by specifying this argument.

Choose one of the following:

- **@follow (-Follow)** — All design objects referenced outside the containment hierarchy of the selected design objects, as well as the internally referenced design objects, are copied and their references are updated to the new location.
- **@nofollow (-NOFollow)** — Design objects referenced outside the containment hierarchy of the selected design objects are not copied and the references to them are not updated. The default setting for reference traversal is off. Default.

- ***preview_mode***

A switch name that specifies whether this function issues a preview report. Choose one of the following:

- **@nopreview (-Nopreview)** — Does not issue the preview report; instead, releases the configuration. Default.
- **@preview (-Preview)** — Issues the preview report only and does not actually perform the copy operation.

- ***copy_mode***

A name that specifies which versions of the design objects to copy. Choose one of the following:

- **@object** — Copies the entire design object. Default.
- **@version** — Copies only the specified version of the design object.

- ***overwrite***

A name that specifies the action taken if conflicting design objects exist in the destination directory. Choose one of the following:

- **@cancel** — Cancels the copy operation. Currently, this value does not work. Default.
- **@replace** — Replaces the conflicting design objects.

- ***create_dest***

A name that specifies the action taken if the destination does not exist. Choose one of the following:

- **@nocreate** — Does not create the target directory and cancels the copy operation. Default.
- **@create** — Creates the target directory and proceeds with the copy operation.

- ***update***

A switch name that specifies whether to update references of the copied object after the copy operation. Choose one of the following:

- **@update** — Updates references after the copy operation. Default.
- **@nouupdate** — Does not update references after the copy operation.

Return Values

VOID.

Description

The toolkit function `$$copy_object()` copies each of the specified design objects to a target location and updates its references to the new location. You must specify the absolute pathname and type for each design object in the `object_list` argument.

The `filter_list` argument enables you to specify which objects are to be excluded from the copy, based on their pathname. The `follow_refs` argument enables you to turn reference traversal on or off.

You can preview the results of this operation before executing it, by specifying the `preview_mode` argument. The results are written to the session configuration area.

The `copy_mode` argument enables you to specify whether the entire object or only a specific version is copied. To copy only a particular version, you specify the version number in the `object_list` and `@version` as the `copy_mode` argument. If you specify a version number in the object list, as well as `@object` as the `copy_mode` argument, this function copies the entire design object.

This function cancels the copy operation if conflicting design objects exist in the destination directory. To replace the conflicting design objects, you choose the `@replace` value of the `overwrite` argument. If you execute this function on a container type object with the `overwrite` argument specified as `@replace` and a target conflict occurs, the conflicting design object is replaced as long as it is also a container type object. The replacement does not occur if either the source or the destination object are not container type objects.

If the destination directory specified in the `target` argument does not exist, by default, this function cancels the copy operation. To create the directory and proceed with the copy, you specify `@create` as the `create_dest` argument. In this case, if you specify a single design object to copy and specify a destination directory that does not exist, this function renames the object with the new leaf name. If you specify multiple design objects to copy and specify a destination directory that does not exist, this function creates the destination directory and copies the design objects into the directory. If you specify a destination directory in which any part of the pathname except for the leaf is non-existent, this function cancels and an error message states that the destination container does not exist.

By default, the `$$copy_object` function updates the references of the copied object at the completion of the copy operation. If you want to prevent the references from updating, use the `@noupdate` switch.

Examples

The following example copies version 2 of design object *plans* and version 3 of the design object *staff*, to the destination directory `$PROJECT_XYZ/d2/budget`. If conflicting design objects exist in the destination directory, the copy is canceled. If the destination directory does not exist, the directory is created. The copied object is not updated after the copy operation.

```
$$copy_object(["$PROJECT_XYZ/d1/plans", "Mgc_file", 2],  
              ["$PROJECT_XYZ/d1/staff", "Mgc_file", 3]],  
              "$PROJECT_XYZ/d2/budget", "", @follow, @noupdate @version,  
              @cancel, @create, @noupdate);
```

Related Topics

[\\$\\$change_object_name\(\)](#)

[\\$\\$duplicate_object\(\)](#)

[\\$\\$move_object\(\)](#)

\$copy_object()

Scope: dmgr_model or dmgr_version_model

Prerequisite: To use this function, you must be in an active Navigator window and must select at least one design object.

Copies the selected design objects to a target location.

Usage

`$copy_object("destination", [filter], follow_refs, preview, copy_mode, overwrite, update)`

COPY Object "destination" [filter] follow_refs preview copy_mode overwrite update

Edit > Copy

Arguments

- **destination**

A string that specifies the absolute or relative pathname of the target location.

- **filter**

A vector of string vectors that specifies which objects are to be excluded from the release, based on their pathname.

`["string1", "string2", "stringN"]`

You can specify the string pattern by entering a literal string or by using UNIX System V regular expressions. The default is to include all objects in the release.

- **follow_refs**

A name that enables you to set reference traversal to either on or off. If the selected object is a container design object, all objects contained within the container are copied and their references are updated. Objects that are referenced by the selected object or by any object contained within its containment may be included or excluded by specifying this argument.

Choose one of the following:

- **@follow (-Follow)** — All design objects referenced outside the containment hierarchy of the selected design objects, as well as internally referenced design objects, are copied and their references are updated to the new location.
- **@nofollow (-NOFollow)** — Design objects referenced outside the containment hierarchy of the selected design objects, are not copied and the references to them are not updated. The default setting for reference traversal is off. Default.

- **preview**

A switch name that specifies whether this function issues a preview report. Choose one of the following:

- **@noprview (-Noprview)** — Does not issue the preview report; instead, performs the copy operation. Default.

- **@preview (-Preview)** — Issues the preview report only and does not perform the copy operation.
- ***copy_mode***

A switch name that specifies which versions of the selected design object to copy. Choose one of the following:

 - **@object (-Object)** — Copies all versions of each selected design object. Default.
 - **@version (-Version)** — Copies only the current version of each selected design object.
- ***overwrite***

A switch name that specifies the action taken if conflicting design objects exist at the target location. Choose one of the following:

 - **@ask_overwrite (-Ask_overwrite)** — Asks for permission to replace the conflicting design object with the selected design object. Default.
 - **@cancel (-Cancel)** — Cancels the copy operation.
 - **@replace (-Replace)** — Replaces the conflicting container type design object with the selected design object.
- ***update***

A switch name that specifies whether to update references of the copied object after the copy operation. Choose one of the following:

 - **@update (-Update)** — Updates references after the copy operation. Default.
 - **@noupdate (-NOUpdate)** — Does not update references after the copy operation.

Return Values

VOID.

Description

The interactive function `$copy_object()` copies the selected design object to the directory specified in the destination argument and updates its references to the new location. You can specify the destination argument as an absolute pathname, a relative pathname, or as a new leaf name. If you select a single design object to copy and specify the destination directory using a new leaf name, this function copies the selected design object to the destination directory and renames the object with the new leaf name.

You can press the Copy key, as a short-cut method for executing the `$copy_object()` function, as noted in “[Key Names Mapped to Functions & Scopes](#)” on page 804.

If you run this function in the Navigator window, you can select and copy several design objects simultaneously. By default, this function copies all versions of each selected design object. To copy only the current version, you must specify `@version` as the `copy_mode` argument. To copy

a previous version of a design object, you use the \$copy_version() function from the version window.

If conflicting design objects exist at the target location, this function asks for permission to replace those objects with the selected design objects. To replace the conflicting design objects, you choose @replace as the overwrite argument. If you execute this function on a container type object with the overwrite argument specified as @replace and a target conflict occurs, the conflicting design object is replaced as long as it is also a container type object. The replacement does not occur if either the source or the destination object are not container type objects. To cancel the copy operation if conflicting objects exist, you choose @cancel.

By default, the \$copy_object function updates the references of the copied object at the completion of the copy operation. If you want to prevent the references from updating, use the @noupdate switch.

Copying an object by using the \$copy_object() function is an interruptible operation. At any time during the execution of this function, you can press the Kill key to interrupt the operation. When you halt the copy operation, a dialog box is displayed prompting you to abort or to continue the operation. If you choose to abort, the operation is halted.

Interrupting the copy operation in this way, halts the operation and the Pyxis Project Manager application deletes any directories created by the execution of this function. In some cases, particularly if the interrupt is executed late in the operation, the clean-up procedures might not successfully remove all the directories created by this function, at the source or destination; however, the source object always remains undamaged.

Examples

1. This example specifies a leaf name for the destination of the copy operation. The function copies the design object with the new leaf name in the current navigator directory and does not update references.

```
$copy_object("new_file", @object, @cancel, @noupdate);
```

2. This example uses a relative pathname and the @ask_overwrite value to copy the current version of the selected design object. The function copies the design object and places it in the directory relative to the current navigator directory and updates references.

```
$copy_object("../", @version, @ask_overwrite, @update);
```

3. This example uses an absolute pathname and the @replace value to copy the selected design object. If a conflicting object exists, this function replaces it and updates references (by default).

```
$copy_object("$PROJECT_XYZ/base", @object, @replace);
```

4. These examples show command syntax from the popup command line.

cop ob new_file -object -cancel -noupdate

cop ob ../ -version -ask_overwrite -update

cop ob \$PROJECT_XYZ/base -object -replace

Related Topics

[\\$change_object_name\(\)](#)

[\\$delete_object\(\)](#)

[\\$copy_version\(\)](#)

[\\$move_object\(\)](#)

\$copy_version()

Scope: dmgr_version_model

Prerequisite: To use this function, you must be in an active version window and must select exactly one design object version.

Copies a single selected version of the design object to a target location.

Usage

`$copy_version("destination", overwrite)`

COPy VErSion destination *overwrite*

Edit > Copy

Versions window popup menu > **Copy**

Arguments

- **destination**

A string that specifies the absolute pathname of the target location.

- **overwrite**

A switch name that specifies the action taken if conflicting design objects exist at the target location.

Choose one of the following:

- **@ask_overwrite (-Ask_overwrite)** — Asks for permission to replace the conflicting design object with the selected design object version. Default.
- **@cancel (-Cancel)** — Cancels the copy operation.
- **@replace (-Replace)** — Replaces the conflicting design object with the selected design object version.

Return Values

VOID.

Description

The interactive function `$copy_version()` copies the selected design object version to the directory specified in the destination argument. You can specify the destination argument as an absolute pathname, a relative pathname, or a new leaf name. If you specify the pathname as a new leaf name, this function creates a copy of the design object in the current navigator directory with the new name.

If conflicting design objects exist at the target location by default, this function asks for permission to replace those objects with the selected design objects. To replace conflicting design objects automatically, you choose `@replace` as the overwrite argument. To cancel the copy operation if conflicting objects exist, you choose `@cancel`.

You can press the Copy key, as a shortcut method for executing the \$copy_version() function, as noted in “[Key Names Mapped to Functions & Scopes](#)” on page 804.

You can interrupt the \$copy_version() operation at any time during its execution by pressing the Kill key. When you halt the copy operation, a dialog box appears, prompting you to abort or to continue the operation. If you choose to abort, the operation is halted.

Interrupting the copy operation in this way halts the operation and attempts to return the configuration to its original state by deleting the directories created by the function. However, these clean-up procedures do not remove all the directories that this function creates at the source or destination. In order to ensure containers that have only one version are not destroyed, only containers that are completely empty are removed.

Examples

1. This example specifies a leaf name for the destination of the copy operation. The function copies the design object version to the new leaf name in the current navigator directory.

```
$copy_version("new_file", @cancel);
```

2. This example uses a relative pathname and the @ask_overwrite value to copy the selected version of the design object. The function copies the design object version and places it in the directory relative to the current navigator directory.

```
$copy_object("../", @ask_overwrite);
```

3. This example uses an absolute pathname and the @replace value to copy the selected version of the design object. If a conflicting object exists, this function replaces it.

```
$copy_object("$PROJECT_XYZ/base", @replace);
```

4. These examples show command syntax from the popup command line.

```
cop ve new_file -cancel
```

```
cop ve ../ -ask_overwrite
```

```
cop ve $PROJECT_XYZ/base -replace
```

Related Topics

[\\$add_versions\(\)](#)

[\\$change_version_depth\(\)](#)

[\\$copy_object\(\)](#)

[\\$\\$delete_version\(\)](#)

[\\$freeze_version\(\)](#)

[\\$revert_version\(\)](#)

[\\$select_version\(\)](#)

[\\$show_versions\(\)](#)

[\\$unfreeze_version\(\)](#)

[\\$unselect_version\(\)](#)

`$report_version_info()`

\$\$create_configuration()

Scope: dm_config_area, dm_config_status_area, dm_config_tk, or dm_config_window

Creates a configuration object.

Usage

```
$$create_configuration("path")
```

Arguments

- **path**

A string that specifies the pathname of the new configuration object. You can specify a relative or absolute pathname. If you specify a relative pathname, the Pyxis Project Manager application assumes that the path is relative to the current working directory. An error is returned if the object specified in this pathname already exists.

Default: "", which is an empty string that creates a new untitled configuration object.

Return Values

VOID.

Description

The toolkit function `$$create_configuration()` creates a configuration object. The new configuration becomes the active configuration. To name the new configuration object, you must specify the path argument.

Examples

1. This example creates a configuration object named *config_a* in the *\$PROJECT_XYZ* directory.

```
$$create_configuration("$PROJECT_XYZ/config_a");
```

2. This example creates an untitled configuration object.

```
$$create_configuration("");
```

Related Topics

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

[\\$\\$save_configuration\(\)](#)

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

[\\$\\$save_configuration_as\(\)](#)

\$create_dm_category()

Scope: dm_proj_tk

Creates a category at the specified path.

Usage

```
$create_dm_category("path");
```

Arguments

- **path**

A string value containing the path at which the category should be created. This is the full path including the name of the category.

Return Values

A Boolean that indicates whether the object was created successfully or not.

Description

Categories may be created within libraries, external libraries, and technology categories. Categories can also be created within other categories, enabling nested hierarchies to an arbitrary depth.

Examples

The following example creates a category “cat” within the library at */tmp/project/lib*.

```
$create_dm_category("/tmp/project/lib/cat");
```

Related Topics

[\\$create_dm_cell\(\)](#)

[\\$create_dm_ext_lib\(\)](#)

[\\$create_dm_library\(\)](#)

[\\$create_dm_project\(\)](#)

[\\$create_dm_tech_category\(\)](#)

[\\$create_dm_tech_lib\(\)](#)

\$create_dm_cell()

Scope: dm_proj_tk

Creates a new cell at the specified path. Cells can be created in libraries, external libraries, categories, and technology categories.

Usage

```
$create_dm_cell("path");
```

Arguments

- **path**

A string value containing the path at which the cell should be created. This is the full path including the name of the cell.

Return Values

A Boolean that indicates whether the object was created successfully or not.

Examples

The following example creates a cell named “cell” in the external library at */tmp/ext_lib*.

```
$create_dm_cell("/tmp/ext_lib/cell");
```

Related Topics

[\\$create_dm_category\(\)](#)

[\\$create_dm_ext_lib\(\)](#)

[\\$create_dm_library\(\)](#)

[\\$create_dm_project\(\)](#)

[\\$create_dm_tech_category\(\)](#)

[\\$create_dm_tech_lib\(\)](#)

\$create_dm_ext_lib()

Scope: dm_proj_tk

Creates an external library at the specified path using the specified technology settings. External libraries may not be created within other typed containers.

Usage

```
$create_dm_ext_lib("path", "tech_lib", "tech_config");
```

Arguments

- **path**
The path at which the new project should be created. This is the full path including the name of the external library.
- **tech_lib**
A string value containing the path to the technology library the external library should use.
- **tech_config**
A string value containing name of the technology configuration the external library should use. This must be the name technology configuration within the technology library specified by the tech_lib argument.

Return Values

A Boolean that indicates whether the object was created successfully or not.

Examples

The following example creates an external library at */tmp/ext_lib* using the technology library */tmp/generic_tech* and technology configuration "generic_65n_5".

```
$create_dm_project("/tmp/ext_lib", "/tmp/generic_tech", "generic_65n_5");
```

Related Topics

[\\$create_dm_category\(\)](#)

[\\$create_dm_cell\(\)](#)

[\\$create_dm_library\(\)](#)

[\\$create_dm_project\(\)](#)

[\\$create_dm_tech_category\(\)](#)

[\\$create_dm_tech_lib\(\)](#)

\$create_dm_library()

Scope: dm_proj_tk

Creates a library at the specified path. Libraries may only be created within projects.

Usage

```
$create_dm_library("path");
```

Arguments

- **path**

A string value containing the path at which the library should be created. This is the full path including the name of the library.

Return Values

A Boolean that indicates whether the object was created successfully or not.

Examples

The following example creates a library “lib” within the project */tmp/project*.

```
$create_dm_library("/tmp/project/lib");
```

Related Topics

[\\$create_dm_category\(\)](#)

[\\$create_dm_cell\(\)](#)

[\\$create_dm_ext_lib\(\)](#)

[\\$create_dm_project\(\)](#)

[\\$create_dm_tech_category\(\)](#)

[\\$create_dm_tech_lib\(\)](#)

\$create_dm_project()

Scope: dm_proj_tk

Creates a new Pyxis Project Manager project at the specified path using the specified technology settings. Projects may not be created within other typed containers.

Usage

```
$create_dm_project("path", "tech_lib", "tech_config");
```

Arguments

- **path**
A string value containing the path at which the new project should be created. This is the full path including the name of the project.
- **tech_lib**
A string value containing the path to the technology library the project should use.
- **tech_config**
A string value containing the name of the technology configuration that the project should use. This must be the name of a technology configuration within the technology library specified by the tech_lib argument.

Return Values

A Boolean that indicates whether the object was created successfully or not.

Examples

The following example creates a project at */tmp/project* using the technology library */tmp/generic_tech* and technology configuration "generic_65n_5".

```
$create_dm_project("/tmp/project", "/tmp/generic_tech", "generic_65n_5");
```

Related Topics

[\\$create_dm_category\(\)](#)

[\\$create_dm_cell\(\)](#)

[\\$create_dm_ext_lib\(\)](#)

[\\$create_dm_library\(\)](#)

[\\$create_dm_tech_category\(\)](#)

[\\$create_dm_tech_lib\(\)](#)

\$create_dm_tech_category()

Scope: dm_proj_tk

Creates a new technology category at the specified path.

Usage

```
$create_dm_tech_category("path");
```

Arguments

- **path**

A string value containing the path at which the technology category should be created. This is the full path including the name of the technology category.

Return Values

A Boolean that indicates whether the object was created successfully or not.

Description

Technology categories can be created within technology libraries or other technology categories enabling for a nested hierarchy of arbitrary depth within a technology library. Technology categories differ from regular categories in that they enable design objects to be created within them directly.

Examples

The following example creates a technology category “process” within the technology library */tmp/generic_kit* and another technology category “rules” within it.

```
$create_dm_tech_category("/tmp/generic_kit/process");
```

```
$create_dm_tech_category("/tmp/generic_kit/process/rules");
```

Related Topics

[\\$create_dm_category\(\)](#)

[\\$create_dm_cell\(\)](#)

[\\$create_dm_ext_lib\(\)](#)

[\\$create_dm_library\(\)](#)

[\\$create_dm_project\(\)](#)

[\\$create_dm_tech_lib\(\)](#)

\$create_dm_tech_lib()

Scope: dm_proj_tk

Creates a new technology library at the specified path. Technology libraries may not be created within other typed containers.

Usage

```
$create_dm_tech_lib("path");
```

Arguments

- **path**

A string value containing the path at which the technology library should be created. This is the full path including the name of the technology library.

Return Values

A Boolean that indicates whether the object was created successfully or not.

Examples

The following example creates a new technology library named “tech_lib” at */tmp/generic_kit*.

```
$create_dm_tech_lib("/tmp/generic_kit");
```

Related Topics

[\\$create_dm_category\(\)](#)

[\\$create_dm_cell\(\)](#)

[\\$create_dm_ext_lib\(\)](#)

[\\$create_dm_library\(\)](#)

[\\$create_dm_project\(\)](#)

[\\$create_dm_tech_lib\(\)](#)

[\\$create_tech_config_object\(\)](#)

\$create_symbols_for_hdl_lib()

Scope: lang_ifc

Given an ADMS or Modelsim compiled library path, this function generates symbols for each design unit in the library, and registers the symbols to their associated models.

Usage

`$create_symbols_for_hdl_lib("destination_path", "lib_name")`

Edit > Language > Register Model > From Compiled Library

Arguments

- **destination_path**
A string value which specifies the path to an existing category, library or external library where new components and symbols are generated.
- **lib_name**
A string value which specifies the name of the library, as used in the project's ini file.

Return Values

VOID.

Description

The compiled library should already be referenced by the current hierarchy's ini file. If it is an external compiled library, the user should have already included it using `$set_ini_mappings()`.

Examples

The following example creates cells and symbols inside the *\$PROJ/libA/catB* category, and registers them to models in the ADMS compiled library named "MY_ADMS_MODELS":

```
$create_symbols_for_hdl_lib("$PROJ/libA/catB", "MY_ADMS_MODELS");
```

Related Topics

[\\$create_symbols_for_src\(\)](#)

\$create_symbols_for_src()

Scope: lang_ifc

Generates symbols for all design units defined in that file or hierarchy.

Usage

```
$create_symbols_for_src("src_path", "src_type")
```

Edit > Language > Register Model > Register Model

Arguments

- **src_path**
A string value specifying the path to the source file or hierarchy.
- **src_type**
A string value specifying the type of the source file or hierarchy design object (such as “mgc_verilog” or “mgc_library”).

Return Values

VOID.

Description

Given a path to a source file, this function generates symbols for all design units defined in that file. Given a path to a hierarchy, it generates symbols for all design units in all source files contained in that hierarchy.

Examples

The following example generates symbols for all design units at the source file location of “mgc_verilog” type:

```
$create_symbols_for_src("$MYDESIGN/inv1/inv1", "mgc_verilog");
```

Related Topics

[\\$create_symbols_for_hdl_lib\(\)](#)

\$create_tech_config_object()

Scope: dm_proj_tk

Creates a new technology configuration within the specified technology library specified by using the input settings.

Usage

```
$create_tech_config_object("name", "tech_lib", "process", "sdl_rules", "lvs_rules",  
    "drc_rules", "pex_rules", "userware_dir");
```

Arguments

- **name**
A string value containing the name of the technology configuration that is to be created.
- **tech_lib**
A string value containing the path to the technology library within which the configuration is to be created. This is the full path to the technology library including its name.
- **process**
A string value containing the path to the ic_persist_process that the technology configuration uses.
- **sdl_rules**
A string value containing the path to the SDL rules file that the technology configuration uses. This argument is optional.
- **lvs_rules**
A string value containing the path to the LVS rules file that the technology configuration uses. This argument is optional.
- **drc_rules**
A string value containing the path to the DRC rules file that the technology configuration uses. This argument is optional.
- **pex_rules**
A string value containing the path to the PEX rules file that the technology configuration uses. This argument is optional.
- **userware_dir**
The top level path to a directory that are added to the AMPLE path when this technology configuration is active within the Pyxis Project Manager application.

Return Values

A Boolean that indicates whether the object was created successfully or not.

Description

With the exception of the process all settings are optional. The files specified for the options must reside within the technology library.

Examples

The following example creates a very basic technology configuration within the technology library */tmp/generic_kit*.

```
$create_tech_config_object(  
  
  "generic_65n_6",  
    "/tmp/generic_kit",  
    "/tmp/generic_kit/process/generic_65n_6",  
    "/tmp/generic_kit/process/rules/SDL",  
    '''  
    "/tmp/generic_kit/userware");
```

Related Topics

[\\$create_dm_tech_lib\(\)](#)

[\\$get_technology\(\)](#)

[\\$create_dm_tech_category\(\)](#)

[\\$set_technology\(\)](#)

\$\$create_versioned_object()

Scope: dme_do_tk

Creates a versioned design object that encapsulates data from a non-Mentor Graphics application.

Usage

```
$$create_versioned_object("obj_name", "obj_type")
```

Arguments

- **obj_name**
A string that specifies the pathname of the versioned design object to be created.
- **obj_type**
A string that specifies the type of the versioned design object to be created.

Return Values

A Boolean that indicates whether the versioned design object was created. If the design object was successfully created, this function returns @true; otherwise, this function returns @false.

Description

The toolkit function `$$create_versioned_object()` creates a versioned design object that encapsulates a non-Mentor Graphics data file, and opens a working copy of the new design object. The specified third-party data file must be a fileset member of a registered design object type.

You should use this function in conjunction with the `$$close_versioned_object()` and `$$open_versioned_object()` functions. This set of functions enable you to apply the Mentor Graphics version management scheme to design objects that encapsulate data from non-Mentor Graphics applications.

Note



Note that the use of the `$$close_versioned_object()`, `$$create_versioned_object()`, and `$$open_versioned_object()` functions is not always required for encapsulating data from non-Mentor Graphics applications. The only situation where these functions are required is for managing *versions* of data from non-Mentor Graphics applications. For information about encapsulating data, refer to the *Pyxis Registrar User's and Reference Manual* on SupportNet.

When you create a versioned design object, this function places a lock on the versioned design object and opens a working copy of that object. You can use the appropriate non-Mentor Graphics application to make changes to the working copy. To close the design object, you must use the `$$close_versioned_object()` function which removes the lock from the object, saves the data, and updates the attributes and version information about the object.

Examples

Assume you have a third-party tool that works on files with the suffix *.hck*, and that the type *Phantom* has a fileset member *.hck*. You have registered the *Phantom* type and the third-party tool in your local registry.

This example creates the “my_file” versioned design object of *Phantom* type.

```
$$create_versioned_object("$PROJECT_XYZ/my_file","Phantom");
```

At this point, the following files exist:

```
my_file.Pantom.lck
```

```
my_file.hck           //Created by third party-tool.
```

Related Topics

[\\$\\$close_versioned_object\(\)](#)

[\\$\\$open_versioned_object\(\)](#)

\$create_work_area_from_rc()

Scope: dmgr_project_model

Populates the specified work area with the specified root hierarchy from the specified server and repository using the specified revision value.

Usage

`$create_work_area_from_rc("server_name", "repository", "root_hdo", "dest_path", "revision")`

Arguments

- **server_name**
A string that specifies the name of the server to use when creating the work area.
- **repository**
A string that specifies the name of the repository to use when creating the work area.
- **dest_path**
A string that specifies the pathname to the work area.
- **revision**
A string that specifies the revision to use when creating the work area. Default: HEAD.

Return Values

VOID.

Description

This function does not perform `$check_objects_from_rc()`. This operation may be performed any number of times.

Examples

```
$create_work_area_from_rc($rcs_default_server(), $rcs_default_repository(),  
"project_name", "/path/to/work/area");
```

```
$create_work_area_from_rc($rcs_default_server(), $rcs_default_repository(),  
"project_name", "/path/to/work/area", "123");
```

Related Topics

[\\$add_project_to_rc\(\)](#)

[\\$cancel_checkout_objects_for_rc\(\)](#)

[\\$checkin_objects_to_rc\(\)](#)

[\\$checkin_objects_to_rc\(\)](#)

[\\$update_objects_from_rc\(\)](#)

[\\$object_status_for_rc\(\)](#)

[\\$revert_object_from_rc\(\)](#)

Chapter 3

Function Dictionary Part 2

Temporary short description for converting multi .fm chapter.

\$\$delete_configuration()

Scope: dm_config_tk

Deletes all design object versions in the active configuration from the disk.

Usage

\$\$delete_configuration()

Arguments

None.

Return Values

VOID.

Description

The \$\$delete_configuration() function deletes from the disk all the design objects in the active configuration, except for container design objects that are not empty. This function only deletes design objects of the type *Mgc_container* if they are completely empty. If a container design object is not deleted because it is not empty, the container is still removed from the configuration but it is not deleted from the disk. Entries only remain in the configuration after a \$\$delete_configuration() operation when an error occurs during its execution.

This function deletes the versions of both retargetable and non-retargetable entries. However, it only deletes the versions of the design object that are actually in the configuration. If a design object with only one version is part of the configuration, this function deletes the entire object from the disk. If a previous version is part of the configuration, only that version of the object is deleted.

Note



After this function runs, the deleted design objects are not recoverable. You should use this function with caution.

This function does not delete the configuration object itself. To delete the actual configuration object, use the design object toolkit function \$\$delete_object().

Examples

This example opens the configuration “config_beta” and then deletes it from the disk.

```
$$open_configuration("$PROJECT_XYZ/config_beta", 0, @read_write);  
  
$$delete_configuration();
```

Related Topics

[\\$\\$freeze_configuration\(\)](#)

[\\$\\$unfreeze_configuration\(\)](#)

\$\$delete_object()

\$delete_configuration()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window.

Deletes all design object versions in the active Configuration window from the disk.

Usage

\$delete_configuration()

DELEte COntfiguration *-force*

Edit > Delete Configuration

Configuration window popup menu > **Global Operations** > **Delete Configuration**

Arguments

- *force (-Force)*

A switch name that enables you to suppress the dialog box. This switch is available only when you execute this function from the command line by using command syntax.

Return Values

VOID.

Description

The interactive function \$delete_configuration() deletes from the disk all design object versions in the active Configuration window, except for container design objects that are not empty. This function only deletes design objects of the type "Mgc_container" if they are completely empty. If a container design object is not deleted because it is not empty, the container is still removed from the configuration but it is not deleted from the disk. Entries only remain in the configuration after a \$delete_configuration() operation when an error occurs during its execution.

This function deletes the versions of both retargetable and non-retargetable entries. However, it only deletes the versions that are actually in the configuration. If a design object with only one version is part of the configuration, this function deletes the entire object from the disk. If a previous version is part of the configuration, only that version of the object is deleted. After this function executes, the deleted design objects are not recoverable. You should use this function with care.

This function does not delete the configuration object itself. To delete the actual configuration object, you return to the navigator and use the \$delete_object() function. You should use the \$delete_configuration() function with care.

The \$delete_configuration() function reports its status as it executes. The summary includes the number of failures, errors, and warnings found during the operation. Additionally, a message appears in the message area, stating whether the copy operation failed, passed, or passed with warnings.

If your session setup values specify to show the monitor window, the configuration monitor window is opened when you execute this function. To hide the monitor window, execute the monitor window's popup menu item **Hide Monitor**. You can redisplay the status summary by executing the Configuration window's popup menu item **Show Monitor**.

The configuration's monitor window is cleared when a new configuration operation executes. If you want to save the status information in a file, you can use the monitor window's popup menu item **Export**.

Deleting design object versions, using the `$delete_configuration()` function, is an interruptible operation. At any time during the execution of this function, you can press the Kill key to interrupt the operation. When you halt the delete operation, a dialog box is displayed prompting you to abort or to continue the operation. If you choose to abort, the operation is halted.

Interrupting the operation in this way, halts the operation but does not return the design object versions to their original state. Any deletions made prior to the interruption are permanent, while those versions that have not been processed prior to the interruption remain in the configuration.

If you select the **Edit > Delete Configuration** menu item, this function brings up a dialog box that asks for permission to proceed. To suppress the dialog box, you execute this function from the command line by using the command syntax.

Examples

1. This example deletes the contents of the active Configuration window.

```
$delete_configuration();
```

2. This example deletes the configuration and suppresses the dialog box by using command syntax from the popup command line.

```
del co -force
```

Related Topics

[\\$freeze_configuration\(\)](#)

[\\$remove_configuration_entry\(\)](#)

[\\$delete_object\(\)](#)

\$delete_design_object()

Scope: dme_base_tk

Deletes the specified design objects.

Usage

`$delete_design_object([objects], check_locked_objects)`

DELeTE DEsign Object [objects] *check_locked_objects*

Edit > Delete > Object

Arguments

- **objects**

A vector of string vectors that specifies the design objects that are to be deleted. The format of this vector of string vectors is:

`[["object1", "type1"], ["object2", "type2"], ..., ["objectN", "typeN"]]`

The object is a string that specifies the absolute pathname of the design object. The type is a string that specifies the type of the design object.

- ***check_locked_objects***

A Boolean that indicates whether a specified design object is deleted if it contains locked design objects.

Choose one of the following options:

- **@check (-Check)** — Does not delete the specified design object if it contains one or more locked design objects. All other specified design objects are deleted. Default.
- **@no_check (-No_check)** — Deletes the specified design object, even if the design object contains locked design objects.

Return Values

VOID.

Description

The iDM interactive function `$delete_design_object()` deletes all of the versions of the design objects specified in the `objects` argument.

The `check_locked_objects` argument specifies whether containers containing locked objects are deleted. When this argument is set to `@check`, the container to be deleted is searched recursively for the presence of locked objects.

If a design object in the `objects` argument is a container that contains locked objects and you specify the `check_locked_objects` argument as `@check`, the container that contains locked objects is not deleted, but all other specified design objects are deleted. An error message reports that the attempt to delete the container containing locked objects failed. If you specify

the `check_locked_objects` argument as `@no_check`, all objects in the `objects` argument are deleted regardless of whether they contain locked objects.

When you execute the `$delete_design_object()` function using either the menu interface or command syntax, the Delete Design Object dialog box displays enabling you to easily enter your argument selections.

The Delete Design Object dialog box contains a dialog navigator that enables you to navigate through your design hierarchy and easily select the design objects that you want to delete. The dialog box also provides an option for specifying whether you want to delete containers that contain locked objects.

Examples

1. This example deletes the “schematic” design object from the test directory. If the schematic design object contains any locked design objects, it is deleted.

```
$delete_design_object(["$PROJECT_X/test_dir/schematic",  
"mgc_schematic"], @no_check);
```

2. This example deletes the “add_det” and “freq_det” components from the test directory. If one of the components contains a locked object, the component containing the locked object is not deleted.

```
$delete_design_object(["$PROJECT_X/test_dir/add_det",  
"mgc_component"], ["$PROJECT_X/test_dir/freq_det",  
"mgc_component"], @check);
```

3. This example attempts to delete the “analysis” directory which contains the locked waveform database file “forces” using the `@check` option. When the forces design object inside analysis is locked and you specify not to delete if there is a locked object in the container, the object is not deleted.

```
$delete_design_object(  
  ["$PROJECT_XYZ/chref_dir/card_reader/add_det/analysis"  
  "Mgc_container"], @check);
```

This example function call transcripts the following error message:

```
// Error: Attempt to delete object containing locked objects  
  (from: DME/Design Management/DN Design Object Tk A4)
```

Related Topics

[\\$\\$delete_object\(\)](#)

\$delete_excess_versions()

Scope: dmgr_version_model

Prerequisite: To use this function, you must be in an active Navigator window and must select at least one object.

Deletes all but a specified number of versions of the selected design object(s).

Usage

\$delete_excess_versions(keep, *mode*)

DELeTe EXCess Versions keep *mode*

Edit > Delete > Excess Versions

Arguments

- **keep**

An integer that specifies the number of versions to preserve from deletion.

- ***mode***

Choose one of the following options:

- **@containment (-Containment)** — Deletes the specified design object and its underlying hierarchy. Default.
- **@object (-Object)** — Deletes only the specified design object without descending hierarchy below the specified object.
- **@reference (-Reference)** — Follows references below the specified design object and also deletes the referenced objects.

Return Values

VOID.

Description

The \$delete_excess_versions() function deletes a particular number of versions by specifying the number of versions you want to preserve. You have the option of deleting the design hierarchy below the selected object, and the option of deleting those design objects referenced by the selected objects.

This function enables you to delete a number of versions simultaneously; however, it does not delete the last version of a design object. To completely remove a design object, you need to return to the navigator and issue the \$delete_object() function.

Examples

1. This example deletes all but the latest version of the selected design object and the hierarchy beneath it.

\$delete_excess_versions(1, @containment);

2. This example deletes all but the last two versions of the selected design object, but preserves all of the hierarchy beneath it.

\$delete_excess_versions(2, @object);

3. This example deletes all but the last two versions of the selected design object, and also deletes all but the last two versions of all objects referenced by the selected design object.

del ex v 2 -r

Related Topics

[\\$\\$delete_version\(\)](#)

[\\$revert_version\(\)](#)

[\\$\\$delete_version\(\)](#)

[\\$freeze_version\(\)](#)

[\\$\\$prune_design_hierarchy\(\)](#)

[\\$show_versions\(\)](#)

[\\$delete_object\(\)](#)

[Versions](#)

\$\$delete_object()

Scope: dme_do_tk

Deletes the specified design object.

Usage

`$$delete_object("obj_name", "type", check_locked_objects)`

Arguments

- **obj_name**

A string that specifies the absolute or relative pathname of the design object to be deleted. Relative pathnames are resolved using the value of the MGC_WD environment variable.

- **type**

A string that specifies the type of the design object to be deleted.

- ***check_locked_objects***

A Boolean that indicates whether an object is deleted if it contains locked objects.

Choose one of the following options:

- **@check (-Check)** — Does not delete the specified design object if it contains one or more locked objects. Default.
- **@no_check (-No_check)** — Deletes the specified design object, even if the design object contains locked objects.

Return Values

VOID.

Description

The toolkit function `$$delete_object()` deletes all of the versions of the design object specified in the `obj_name` argument. To delete only a specific version of a design object, you use the `$$delete_version()` function.

Deleting a design object using the `$$delete_object()` function is a permanent deletion.

The `check_locked_objects` argument specifies whether containers containing locked design objects are deleted. If a design object in the `obj_name` argument is a container that contains locked design objects and you specify the `check_locked_objects` argument as `@check`, the container that contains locked design objects is not deleted, but all other design objects are deleted. If you specify the `check_locked_objects` argument as `@no_check`, all design objects in the `obj_name` argument are deleted regardless of whether they contain locked design objects.

Examples

This example deletes the primitive “OUT” from the library component “conn_m”. The `check_locked_objects` argument is not specified as the OUT design object is a primitive and cannot contain other design objects.

```
$$delete_object("$DPM_RLSLIB/parts/connectors/conn_m/OUT",  
               "mgc_symbol");
```

Related Topics

[\\$\\$delete_version\(\)](#)

[\\$\\$salvage_object\(\)](#)

\$delete_object()

Scope: dmgr_model

Prerequisite: To use this function, you must be in an active Navigator window and must select at least one design object.

Deletes the selected design object.

Usage

\$delete_object()

DELeTe OBject *-force*

Edit > Delete > Object

Arguments

- *force (-Force)*

A switch name that enables you to suppress the dialog box. This switch is available only when you execute this function from the command line by using command syntax.

Return Values

VOID.

Description

The interactive function \$delete_object() deletes all versions of the selected design object. This function enables you to select and delete several design objects simultaneously. To delete a specific version of a design object, you use the \$\$delete_version() function.

If you select the **Edit > Delete > Object** menu item, this function invokes a dialog box that asks for permission to continue. To suppress the dialog box, run this function from the command line by using the appropriate command syntax.

To undelete design objects, do not use the \$delete_object() function to delete them. The \$delete_object() function deletes design objects permanently. Instead, delete them by dragging them to the trash can. You can retrieve deleted design objects from the trash can by removing them prior to emptying it.

Examples

1. This example deletes the selected design object.

```
$delete_object();
```

2. This example deletes the selected design object and suppresses the dialog box, by using command syntax from the popup command line.

```
del ob -force
```

Related Topics

[\\$empty_trash\(\)](#)

[\\$move_object\(\)](#)

[\\$open_trash_window\(\)](#)

[\\$\\$delete_version\(\)](#)

\$\$delete_object_property()

Scope: `dme_do_tk`

Deletes the specified property from the specified design object.

Usage

`$$delete_object_property("obj_name", "obj_type", version, "prop_name")`

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **obj_type**
A string that specifies the type of the design object.
- ***version***
An integer that specifies the version of the design object. The default is the current version.
- **prop_name**
A string that specifies the name of the property to be deleted.

Return Values

VOID.

Description

The toolkit function `$$delete_object_property()` deletes the specified property from the specified design object.

When you set an object property, it remains on the current version of the design object. The object property carries forward as the design object evolves and its version number is incremented. By default, this function deletes the property from the current version of the design object.

If you set an object property on a previous version, that property remains with that version and does not carry forward as the design object evolves. To delete an object property on a previous version, you must specify the version argument.

To delete an object property, you must have permission to edit the selected design object. You can use the `$change_protection()` function to change the protection status of design objects.

Examples

This example deletes the object property "designer" from version 3 of the design object *base*. Deleting this property from version 3 has no effect on earlier or later versions of this design object.

```
$$delete_object_property("$PROJECT_XYZ/d1/base", "Image_file", 3,  
    "designer");
```

Related Topics

[\\$\\$get_object_properties\(\)](#)

[\\$\\$has_object_property\(\)](#)

[\\$\\$get_object_property_value\(\)](#)

[\\$\\$set_object_property\(\)](#)

[\\$change_protection\(\)](#)

\$delete_object_property()

Scope: dmgr_model

Prerequisite: To use this function, you must be in an active Navigator window and must select at least one design object.

Deletes the specified property from the selected design object.

Usage

```
$delete_object_property("prop_name")
```

DELeTe OBject Property prop_name

Edit > Delete > Property

Arguments

- **prop_name**

A string that specifies the name of the property that this function deletes.

Return Values

VOID.

Description

The interactive function \$delete_object_property() deletes the specified property from the current version of the selected design object. This function can delete an object property from more than one design object simultaneously.

To delete a property, you must have permission to edit the selected design object. You can use the \$change_protection() function to change the protection status of design objects. Deleting a property from a design object does not create a new version of that design object.

Caution



This function does not ask you to confirm the deletion. Instead, it simply deletes the property from the selected design object.

Examples

1. This example deletes the object property "test_case" from each of the selected design objects.

```
$delete_object_property("test_case");
```

2. This example deletes the same property by using command syntax from the popup command line.

```
del ob p test_case
```


Related Topics

[`\$add_object_property\(\)`](#)

[`\$change_protection\(\)`](#)

[`\$report_object_info\(\)`](#)

[Design Object Properties](#)

\$\$delete_reference()

Scope: dme_do_tk

Deletes the specified reference from the specified source design object.

Usage

```
$$delete_reference("obj_name", "obj_type", object_version, "target_obj_name", "target_type",  
                  target_version)
```

Arguments

- **obj_name**
A string that specifies the pathname of the source design object.
- **obj_type**
A string that specifies the type of the source design object.
- ***object_version***
An integer that specifies the version number of the source design object. The default is the current version.
- **target_obj_name**
A string that specifies the pathname of the target design object of the reference to be deleted.
- **target_type**
A string that specifies the type of the target design object of the reference to be deleted.
- ***target_version***
An integer that specifies the version number of the target design object. The default is the current version.

Return Values

VOID.

Description

The toolkit function `$$delete_reference()` deletes from the specified design object all instances of the specified reference.

To delete a reference from a previous version of the source design object, you must specify the `object_version` argument. Deleting a reference from a previous version has no effect on other versions of the source design object.

As other tools can create duplicate references, you can use the `$$delete_reference_handle()` function for more accuracy when deleting references. To find a reference's target handle, you use the `$$get_object_references()` function.

If you specify the `target_version` argument, this function deletes all of the fixed references whose name, type, and version number match the specified arguments. If you omit the

target_version argument, then this function deletes all of the current references found whose name and type match the specified arguments.

This function can only delete references that hold the reference property “creating_tool”, whose value is “project manager”. When you add a reference by using the \$\$add_reference() function, the creating_tool property is automatically added to the reference.

To delete references, you must have permission to edit the specified source design object. You can use the \$change_protection() function to change the protection status of design objects. To change a design object's references, you use the configuration management function \$\$change_object_references().

Examples

1. This example deletes from the source design object, *model*, all references to version 3 of the target design object, *tests*. As the target_version argument is specified, the state of this reference is fixed.

```
$$delete_reference("$PROJECT_XYZ/model", "Image_file", ,  
"$PROJECT_XYZ/d1/tests", "Document_file", 3);
```

2. This example deletes from the source design object, *model*, all references to the target design object, *tests*. As the target_version argument is not specified, the state of this reference is current.

```
$$delete_reference("$PROJECT_XYZ/model", "Image_file", ,  
"$PROJECT_XYZ/d1/tests", "Document_file",,);
```

3. This example assumes that the current version of the design object, *model*, is 5. A reference is deleted from version 3 of *model*, which has no effect on earlier or later versions of the source design object.

```
$$delete_reference("$PROJECT_XYZ/model", "Image_file", 3,  
"$PROJECT_XYZ/d1/tests", "Document_file",,);
```

Related Topics

[\\$\\$add_reference\(\)](#)

[\\$\\$delete_reference_handle\(\)](#)

[\\$change_protection\(\)](#)

[\\$\\$change_object_references\(\)](#)

[\\$\\$get_object_references\(\)](#)

[\\$\\$change_object_references\(\)](#)

\$delete_reference()

Scope: dmgr_reference_model

Prerequisite: To use this function, you must first select a design object and issue the \$show_references() function to create a Reference window. Then, you must be in an active Reference window and select at least one reference.

Deletes the selected reference from the source design object.

Usage

\$delete_reference()

DELEte REference *-force*

Edit > Delete Reference

References window popup menu > **Delete**

Arguments

- *force (-Force)*

A switch name that enables you to suppress the dialog box. This switch is available only when you execute this function from the command line by using command syntax.

Return Values

VOID.

Description

The interactive function \$delete_reference() deletes the selected reference. This function can only delete references from the current version of the selected source design object. You cannot delete references from previous versions of a design object. Deleting references from a design object does not create a new version of the source design object.

Deleting a reference does not delete the source design object or the target design object. Instead, it only deletes the pointer from the source design object to the target design object.

When you select the **Edit > Delete Reference** menu item, this function brings up a dialog box that asks for permission to proceed. To suppress the dialog box execute this function from the command line by using command syntax.

To delete references, you must have permission to edit the selected design object. You can use the \$change_protection() function to change the protection status of design objects.

Note that this function is not available in the dmgr_model scope, which is created when you select a design object in a Navigator window and click on the right arrow to issue the \$explore_references() function. The \$delete_reference() function is available only in an active Reference window, which is created by the \$show_references() function.

Examples

1. This example deletes the selected reference.

```
$delete_reference();
```

2. This example deletes the selected reference and suppresses the dialog box by using command syntax from the popup command line.

```
del re -force
```

Related Topics

[\\$add_reference\(\)](#)

[\\$show_references\(\)](#)

[\\$report_reference_info\(\)](#)

[\\$change_protection\(\)](#)

\$\$delete_reference_handle()

Scope: dme_do_tk

Deletes a reference from the specified design object by using the reference target handle.

Usage

`$$delete_reference_handle("obj_name", "obj_type", obj_version, target_handle)`

Arguments

- **obj_name**
A string that specifies the pathname of the source design object.
- **obj_type**
A string that specifies the type of the source design object.
- **obj_version**
An integer that specifies the version number of the source design object. The default is the current version.
- **target_handle**
An integer that specifies the handle of the target design object.

Return Values

VOID.

Description

The toolkit function `$$delete_reference_handle()` deletes a reference of the specified design object by using the reference target handle. Deleting a reference by using the target handle enables you to specify distinctly which reference to delete. To find the reference's target handle, you use the `$$get_object_references()` function.

To delete a reference from a previous version of the source design object, you must specify the `obj_version` argument. Deleting a reference from a previous version has no effect on other versions of the source design object.

This function can only delete references that hold the reference property "creating_tool", whose value is "project manager". When you add a reference by using the `$$add_reference()` function, the `creating_tool` property is automatically added to the reference.

To delete references, you must have permission to edit the specified source design object. You can use the `$change_protection()` function to change the protection status of design objects. To change a design object's references, you use the configuration management function `$$change_object_references()`.

Examples

1. This example deletes the reference from the current version of the design object *model* by using the target reference handle 3.

```
$$delete_reference_handle("$PROJECT_XYZ/parts/model",  
"Document_file", , 3);
```

2. In this example assume that the current version of the design object *model* is 5. A reference is deleted from version 4 of *model*, which has no effect on earlier or later versions.

```
$$delete_reference_handle("$PROJECT_XYZ/parts/model",  
"Document_file", 4, 3);
```

Related Topics

[\\$\\$add_reference\(\)](#)

[\\$\\$delete_reference\(\)](#)

[\\$\\$change_object_references\(\)](#)

[\\$\\$get_object_references\(\)](#)

[\\$change_protection\(\)](#)

\$\$delete_reference_property()

Scope: dme_do_tk

Deletes a property from the specified reference.

Usage

```
$$delete_reference_property("obj_name", "obj_type", obj_version, "target_obj_name",  
    "target_type", target_version, "prop_name")
```

Arguments

- **obj_name**
A string that specifies the pathname of the source design object.
- **obj_type**
A string that specifies the type of the source design object.
- ***obj_version***
An integer that specifies the version number of the source design object. The default is the current version.
- **target_obj_name**
A string that specifies the pathname of the target design object.
- **target_type**
A string that specifies the type of the target design object.
- ***target_version***
An integer that specifies the version number of the target design object. The default is the current version.
- **prop_name**
A string that is name of the property to be deleted.

Return Values

VOID.

Description

The toolkit function `$$delete_reference_property()` deletes all instances of the specified property from the specified reference. To delete a reference property from a previous version of the source design object, you specify the `obj_version` argument. This function has no effect on other versions of the source design object.

As other tools can create duplicate references, you can use the `$$delete_reference_property_handle()` function for more accuracy when deleting reference properties. To find a reference's target handle, use the `$$get_object_references()` function.

If you specify the `target_version` argument, this function deletes the property from *all* of the fixed references found whose name, type, property, and version number match the specified arguments.

If you omit the `target_version` argument, then this function deletes the property from *all* of the current references found whose name, type, and property, match the specified arguments.

This function can only delete reference properties from references that hold the property “`creating_tool`”, whose value is “`project manager`”. When you add a reference by using the `$$add_reference()` function, the `creating_tool` property is automatically added to the reference.

To delete reference properties, you must have permission to edit the specified source design object. You can use the `$change_protection()` function to change the protection status of design objects.

Examples

1. This example deletes the reference property “`engineer`”. As the `target_version` argument is specified, the reference is fixed and the version number is checked.

```
$$delete_reference_property("$PROJECT_XYZ/amino_acids",  
    "Document_file", , "$PROJECT_XYZ/example/sheet", "Dss_sheet", 3,  
    "engineer");
```

2. This example deletes the reference property “`engineer`”. As the `target_version` argument not specified, the reference is current and the version number is not checked.

```
$$delete_reference_property("$PROJECT_XYZ/amino_acids",  
    "Document_file", , "$PROJECT_XYZ/example/sheet", "Dss_sheet", ,  
    "engineer");
```

3. This example deletes the reference property “`engineer`” from a previous version of the source design object. This deletion has no effect on earlier or later versions of the source design object.

```
$$delete_reference_property("$PROJECT_XYZ/amino_acids",  
    "Document_file", 3, "$PROJECT_XYZ/example/sheet", "Dss_sheet", ,  
    "engineer");
```

Related Topics

[\\$\\$delete_reference_property_handle\(\)](#)

[\\$\\$has_reference_property\(\)](#)

[\\$\\$get_reference_properties\(\)](#)

[\\$\\$set_reference_property\(\)](#)

[\\$\\$get_object_references\(\)](#)

[\\$\\$add_reference\(\)](#)

[\\$change_protection\(\)](#)

\$delete_reference_property()

Scope: dmgr_reference_model

Prerequisite: To use this function, you must first select a design object and issue the \$show_references() function to create a Reference window. Then, you must be in an active Reference window and select at least one reference.

Deletes the specified property from the selected reference.

Usage

`$delete_reference_property("prop_name")`

DELEte REference Property prop_name

Edit > Delete Reference Property

References window popup menu > **Reference Properties > Delete**

Arguments

- **prop_name**

A string that specifies the name of the property to be deleted. This string must not contain any blank spaces.

Return Values

VOID.

Description

The interactive function \$delete_reference_property() deletes the specified property from the current version of the selected reference. This function enables you to delete a reference property from more than one reference simultaneously. Deleting a reference property of a design object does not create a new version of the source or target design object.

To delete a reference property, you must have permission to edit the source design object. You can use the \$change_protection() function to change the protection status of design objects.

Examples

1. This example deletes the reference property "test_eng" from the selected references.

```
$delete_reference_property("test_eng");
```

2. This example deletes the same property by using command syntax from the popup command line.

```
del re p test_eng
```

Related Topics

[`\$add_reference_property\(\)`](#)

[`\$report_reference_info\(\)`](#)

[`\$change_protection\(\)`](#)

\$\$delete_reference_property_handle()

Scope: dme_do_tk

Deletes a property from the specified reference by using the reference target handle.

Usage

```
$$delete_reference_property_handle("obj_name", "obj_type", obj_version, target_handle,  
    "prop_name")
```

Arguments

- **obj_name**
A string that specifies the pathname of the source design object.
- **obj_type**
A string that specifies the type of the source design object.
- **obj_version**
An integer that specifies the version number of the source design object.
- **target_handle**
An integer that specifies the handle of the target design object.
- **prop_name**
A string that specifies name of the property to be deleted.

Return Values

VOID.

Description

The toolkit function `$$delete_reference_property_handle()` deletes the specified reference property by using the target handle, which enables you to specify distinctly which reference property to delete. To find the reference's target handle, you use the `$$get_object_references()` function.

To delete a reference property from a previous version of the source design object, you must specify the `obj_version` argument. Deleting a reference property from a previous version has no effect on other versions of the source design object.

This function can only edit references that hold the reference property "creating_tool", whose value is "project manager". When you add a reference by using the `$$add_reference()` function, the `creating_tool` property is automatically added to the reference.

To delete reference properties, you must have permission to edit the specified source design object. You can use the `$change_protection()` function to change the protection status of design objects.

Examples

This example deletes the reference property “engineer” from version 3 of the design object “acids”, by using the target handle 4.

```
$$delete_reference_property_handle("$PROJECT_XYZ/acids",  
    "Document_file", 3, 4, "engineer");
```

Related Topics

[\\$\\$delete_reference_property\(\)](#)

[\\$\\$has_reference_property\(\)](#)

[\\$\\$get_reference_properties\(\)](#)

[\\$\\$set_reference_property\(\)](#)

[\\$\\$get_object_references\(\)](#)

[\\$\\$add_reference\(\)](#)

[\\$change_protection\(\)](#)

\$\$delete_version()

Scope: dme_do_tk

Deletes the specified version of the specified design object.

Usage

```
$$delete_version("obj_name", "obj_type", version)
```

Arguments

- **obj_name**
A string that specifies the pathname of the design object to be deleted.
- **obj_type**
A string that specifies the type of the design object to be deleted.
- **version**
An integer that specifies the version number of the design object to be deleted.

Return Values

VOID.

Description

The toolkit function `$$delete_version()` deletes the specified version of the specified design object. This function cannot delete frozen versions or the current version of a design object.

Examples

This example deletes version 4 of the “list” design object.

```
$$delete_version("$PROJECT_XYZ/project/list", "Document_file", 4);
```

Related Topics

[\\$\\$delete_object\(\)](#)

[\\$\\$revert_version\(\)](#)

[\\$\\$freeze_version\(\)](#)

[\\$\\$set_version_depth\(\)](#)

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

[\\$\\$unfreeze_version\(\)](#)

[\\$\\$get_version_depth\(\)](#)

\$delete_version()

Scope: dmgr_version_model

Prerequisite: To use this function, you must first select a versioned object and issue the \$show_versions() function to create a Versions window. Then, you must be in an active Versions window and select at least one version.

Deletes the selected design object version.

Usage

\$delete_version()

DELeTe VErSion *force*

Edit > Delete

Versions window popup menu > **Delete**

Arguments

- *force (-Force)*

A switch name that enables you to suppress the warning message. This switch is only available when you execute this function from the command line, by using command syntax.

Return Values

VOID.

Description

The interactive function \$delete_version() deletes the selected version. This function enables you to select and delete several versions simultaneously; however, it does not delete the last version of a design object. To completely remove a design object, you need to return to the navigator and issue the \$delete_object() function.

If you select the **Edit > Delete** menu item, this function invokes a dialog box that asks for permission to proceed with the deletion. To suppress the dialog box, run this function from the command line by using the appropriate command syntax.

Examples

1. This example deletes the selected design object version.

```
$delete_version();
```

2. This example deletes the selected design object version and suppresses the dialog box, by using command syntax from the popup command line.

```
del ve -force
```

Related Topics

[\\$delete_object\(\)](#)

[\\$show_versions\(\)](#)

\$\$delete_version_property()

Scope: dme_do_tk

Deletes a property from the specified version.

Usage

```
$$delete_version_property("obj_name", "obj_type", version, "prop_name")
```

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **obj_type**
A string that specifies the type of the design object.
- ***version***
An integer that specifies the version number of the design object. The default is the current version.
- **prop_name**
A string that specifies the name of the property to be deleted.

Return Values

VOID.

Description

The toolkit function `$$delete_version_property()` deletes the specified property from the specified version.

Version properties always remain with the version to which they were originally set. As the design object evolves, a version property does not carry forward to the next version.

By default, this function deletes the property from the current version. To delete a version property from a previous version, you must specify the version argument. Deleting a version property from a previous version has no effect on other versions of the design object.

Examples

This example deletes the version property "test_date" from version 3 of the design object "amino_acids".

```
$$delete_version_property("$PROJECT_XYZ/amino_acids", "Document_file",  
3, "test_date");
```

Related Topics

[\\$\\$get_version_properties\(\)](#)

[\\$\\$set_version_property\(\)](#)

\$descend_hierarchy_one_level()

Scope: dme_hierarchy

Prerequisite: You must have exactly one component selected in the Component Hierarchy window.

Displays the next level of hierarchy directly beneath the selected component in a component hierarchy window.

Usage

\$descend_hierarchy_one_level()

DEScend HIerarchy One Level

Component window (popup menu) > Show Levels > Show 1 Level

Arguments

None.

Return Values

A vector of string vectors that specifies the component hierarchy window name and its entries:

```
[["hierarchy_window_name", ["entry1", entry1_level, "entry1_pathname",  
    "entry1_leafname"], ["entry2", entry2_level, "entry2_pathname", "entry2_leafname"],  
    ..., ["entryN", entryN_level, "entryN_pathname", "entryN_leafname"]]
```

The `hierarchy_window_name` is a string that specifies the window name of the component hierarchy window within the calling session. Each sub-vector consists of an entry name, an `entry_level`, an `entry_pathname`, and an `entry_leafname`.

The `entry_name` is a string that specifies the name of a single entry in the component hierarchy window. Each `entry_level` is an integer that specifies the level at which the corresponding `entry_name` resides in the hierarchy. Each `entry_pathname` is a string that represents the absolute pathname of the `entry_name`. Each `entry_leafname` is a string that designates the leaf name of the corresponding `entry_name`.

Description

The iDM interactive function `$descend_hierarchy_one_level()` displays the design hierarchy of the selected component in a new component hierarchy window. Run this function from the component hierarchy window after selecting a single component.

If you specified exclusion filters the first time that you executed this function in the current session, the filters are used for displaying the new component hierarchy window. If you want to display the new hierarchy listing with new filter settings, use the `$descend_hierarchy_specify_level()` function that enables you to specify new filters or add filters to ones previously specified. Filters always apply to sub-levels of the hierarchy; that is, you can never filter out the root component.

Examples

1. This example executes the function using function syntax.

```
$descend_hierarchy_one_level();
```

2. This example executes the function using command syntax from the popup command line.

```
des hi o l
```

Related Topics

[\\$descend_hierarchy_specify_level\(\)](#)

[\\$show_component_hierarchy\(\)](#)

\$descend_hierarchy_specify_level()

Scope: dme_hierarchy

Prerequisite: You must have exactly one component selected in the component hierarchy window.

Displays the design hierarchy for a selected component in a component hierarchy window using specified level and filter settings.

Usage

`$descend_hierarchy_specify_level(descend_level, [descend_filters], name_display)`

DEScend HIerarchy Specify Level *descend_level descend_filters name_display*

Arguments

- **descend_level**

An integer that specifies the level at which hierarchical traversal stops. The default is level 2 which displays the selected component design object and the level directly beneath the component.

- **descend_filters**

A vector of strings that specifies patterns that are used to filter component design objects from the component hierarchy window. If any portion of a component design object's pathname matches a string in this vector, the component is excluded from the component hierarchy window's display. You can specify the strings in this vector using System V regular expressions.

- **name_display**

A name that specifies whether to display the component's hierarchy using each subcomponent's absolute pathname or using only a leaf name. Options include:

- **@full (-Full)** — The component's design hierarchy is displayed using the component's absolute pathname.
- **@leaf (-Leaf)** — The component's design hierarchy is displayed using the subcomponent's leaf name only. Default.

Return Values

A vector of string vectors that specifies the component hierarchy window name and its entries:

```
[“hierarchy_window_name”, [“entry1”, entry1_level, “entry1_pathname”,
“entry1_leafname”], [“entry2”, entry2_level, “entry2_pathname”, “entry2_leafname”],
..., [“entryN”, entryN_level, “entryN_pathname”, “entryN_leafname”]]
```

The `hierarchy_window_name` is a string that specifies the window name of the component hierarchy window within the calling session. Each sub-vector consists of an entry name, an `entry_level`, an `entry_pathname`, and an `entry_leafname`.

The `entry_name` is a string that specifies the name of a single entry in the component hierarchy window. Each `entry_level` is an integer that specifies the level at which the corresponding `entry_name` resides in the hierarchy. Each `entry_pathname` is a string that represents the absolute pathname of the `entry_name`. Each `entry_leafname` is a string that designates the leaf name of the corresponding `entry_name`.

Description

The iDM interactive function `$descend_hierarchy_specify_level()` displays the design hierarchy for a selected component in the component hierarchy window using specified level and filter settings.

You can specify filters for hierarchy traversal when you execute the `$show_hierarchy()` and `$descend_hierarchy_specify_level()` functions. If you have specified hierarchy traversal filters using one of these functions, the specified filters are displayed the next time that you execute the `$descend_hierarchy_specify_level()` function. You can modify, add to, or delete these filters.

If you want to descend a single level in a component's hierarchy using the filters that you have previously specified, you can use the `$descend_hierarchy_one_level()` function.

Examples

1. This example descends into the *connector* component's hierarchy and displays all its levels without any filters specified.

```
$descend_hierarchy_specify_level(0, [], @leaf);
```

This example displays the following design hierarchy:

```
connector (schm:schematic)
  mrip (primitive)
  gnd (primitive)
  conn_m (primitive)
  outxhier (primitive)
  .vcc (primitive)
```

2. This example descends into the same component using command syntax from the popup command line. All hierarchical levels are displayed and filters are set to exclude *vcc* and *gnd* components.

```
des hi s l 0 ["vcc", "gnd"] -l
```

This example displays the following design hierarchy:

```
connector (schm:schematic)
  mrip (primitive)
  conn_m (primitive)
  outxhier (primitive)
```

Related Topics

[\\$descend_hierarchy_one_level\(\)](#)

[\\$show_component_hierarchy\(\)](#)

\$\$duplicate_object()

Scope: dm_config_tk

Duplicates a design object.

Usage

```
$$duplicate_object("new_name", "name", "type", [filter_list], follow_refs, preview_mode,
    version, overwrite)
```

Arguments

- **new_name**
 A string that specifies the new name of the duplicated design object. You can specify either a leaf name or the absolute pathname of the design object.
- **name**
 A string that specifies the absolute pathname of the design object to be duplicated.
- **type**
 A string that specifies the type of the design object to be duplicated.
- **filter_list**
 A vector of string vectors that specifies which objects are to be excluded from the release, based on their pathname. You can specify the pathname pattern by entering the literal string or by using UNIX System V regular expressions. The default is to include all objects in the release.
- **follow_refs**
 A name that enables you to set reference traversal to either on or off. By default, containment traversal is always set to on, which means that a design object's contained objects are added to the configuration during a build.
 Choose one of the following:
 - **@follow (-Follow)** — All externally referenced design objects of the selected design objects, as well as internally referenced design objects, are included in the released object during the release.
 - **@nofollow (-NOFollow)** — Externally referenced design objects of the selected design objects, are not included in the released object during the release. The default setting for reference traversal is off. Default.
- **preview_mode**
 A switch name that specifies whether this function issues a preview report. Choose one of the following:
 - **@nopreview (-Nopreview)** — Does not issue the preview report; instead, releases the configuration. Default.

- **@preview (-Preview)** — Issues the preview report only and does not release the configuration.
- **version**
An integer that specifies the version number of the design object to be duplicated. The default is 0, which means the entire design object is duplicated.
- **overwrite**
A name that specifies the action taken if a conflicting design object exists. Choose one of the following:
 - **@cancel** — Cancels the duplication operation. Default.
 - **@replace** — Replaces the conflicting container type design objects.

Return Values

VOID.

Description

The toolkit function `$$duplicate_object()` creates a duplicate of the specified design object. If you specify only a leaf name for the `new_name` argument, this function places the new design object in the same directory as the design object being duplicated. If you specify an absolute pathname for the `new_name` argument, this function places the design object at the pathname that you specify.

By default, this function duplicates the entire design object. To duplicate only a particular version, you must specify the `version` argument.

This function duplicates all objects in the specified design object. To exclude objects from the duplication operation, specify the `filter_list` argument.

This function does not traverse references in the design object, by default. To turn reference traversal on, specify the `follow_refs` argument.

You can preview the results of the duplication operation before executing it, by specifying the `preview_mode` argument. The results are written to the session configuration area.

If a conflicting design object exists, this function cancels the duplication operation. To replace the conflicting design object, you must specify `@replace` as the `overwrite` argument.

If you execute this function on a container type object with the `overwrite` argument specified as `@replace` and a target conflict occurs, the conflicting design object is replaced as long as it is also a container type object. The replacement does not occur if either the source or the destination object are not container type objects.

This function automatically updates the references between duplicated design objects.

Examples

1. This example creates a duplicate design object in the same directory. The design object being duplicated is version 2 of *plans*. The name of the new design object is *goals*. All

objects are included in the duplication. References are not followed. The operation is executed without being previewed. If a conflicting design object exists, it is replaced with the new design object.

```
$$duplicate_object("goals", "$PROJECT_XYZ/plans",  
"Document_file", 2, [], @nofollow, @preview, @replace);
```

2. This example duplicates a design object and places it in another directory. The design object being duplicated is *schedule*. The name of the new design object is *goals*. All objects are included in the duplication. References are not followed. The operation is executed without being previewed. If a conflicting design object exists, the duplication is canceled.

```
$$duplicate_object("$PROJECT_ABC/goals",  
"$PROJECT_XYZ/schedule", "Mgc_file", 0, [], @nofollow,  
@preview, @cancel);
```

Related Topics

[\\$\\$change_object_name\(\)](#)

[\\$\\$move_object\(\)](#)

[\\$\\$copy_object\(\)](#)

\$edit_file()

Scope: dmgr_session_window

Prerequisite: You can use this function from any Pyxis Project Manager window.

Opens a file in edit or read-only mode.

Usage

`$edit_file("object_name", "object_type", edit_mode)`

EDIt File *object_name* *object_type* *edit_mode*

Arguments

- **object_name**
A string that specifies the absolute pathname of the file to be edited.
- **object_type**
A string that specifies the type of the file to be edited.
- ***edit_mode***
A switch name that specifies whether the file is opened in edit or read-only mode. Options include:
 - **@edit (-Edit)** — Opens the file in edit mode.
 - **@read_only (-Read_only)** — Opens the file in read-only mode.

Return Values

VOID.

Description

The interactive function `$edit_file()` opens a file in edit or read-only mode. Both the Edit and Read keys call this function. The default editor checks the protection status of the file. If you do not have write permission, the file opens in read-only mode. If you do not have read or write permission, the file is not opened. You can use the `$change_protection()` function to change the protection status of design objects.

When you press the Read key, the `$edit_file()` function executes in read-only mode. If you press the Read key with either zero or multiple objects selected, a dialog navigator displays enabling you to select the object you want to read. If you press the Read key with a single object of the type "Mgc_file" selected, the read-only editor invokes on that object. If you press the Read key with a single object of some type *other* than Mgc_file selected, the Pyxis Project Manager application determines the default tool for that object's type. If the default tool for the object is "Editor", the read-only editor invokes on the object. If the default tool is not Editor, then the following error message is displayed:

```
Unable to read design object. Default tool must be 'Editor'.
```

Examples

1. This example opens a file *review* in edit mode.

```
$edit_file("$PROJECT_XYZ/design/review", "Mgc_file", @edit);
```

2. This example opens the file *review* in read_only mode, by using command syntax on the popup command line.

```
edi fi $PROJECT_XYZ/design/review Mgc_file -read_only
```

Related Topics

[\\$open_object\(\)](#)

[\\$change_protection\(\)](#)

\$empty_trash()

Scope: dmgr_session_window

Prerequisite: You can use this function from the Navigator or Trash window.

Deletes the contents of the Trash window.

Usage

\$empty_trash()

EMPTy TRash

Edit > Empty Trash

Arguments

None.

Return Values

VOID.

Description

The interactive function \$empty_trash() deletes the contents of the Trash window. After you empty the trash, deleted design objects are not recoverable. You can remove a design object from the Trash window by dragging its icon to another window or by the \$untrash_object() function.

When you close the Pyxis Project Manager session, any design objects in the Trash window are deleted without warning.

Caution



The \$empty_trash() function does not ask you to confirm the deletion. Instead, it simply deletes the contents of the Trash window. When you close the Pyxis Project Manager session, all design objects in the Trash window are deleted without warning.

Examples

1. This example empties the contents of the Trash window.

```
$empty_trash();
```

2. This example empties the Trash window by using command syntax from the popup command line.

```
emp tr
```

Related Topics

[\\$delete_object\(\)](#)

[\\$untrash_object\(\)](#)

\$explore_contents()

Scope: dmgr_model

Prerequisite: To use this function, you must be in an active Navigator window and must select exactly one container object or directory.

Navigates down one level of the containment hierarchy.

Usage

\$explore_contents()

EXPlore COnents

Project Navigator window popup menu > **Open** > **Explore Contents**

Navigator window > **Explore** > **Contents**

Arguments

None.

Return Values

VOID.

Description

The interactive function \$explore_contents() navigates down one level of the containment hierarchy and displays the design objects contained within the selected container design object or directory in the Navigator window.

Examples

1. This example explores the contents of the selected container or directory.

```
$explore_contents();
```

2. This example explores the contents of the selected container or directory by using command syntax from the popup command line.

```
exp co
```

Related Topics

[\\$explore_parent\(\)](#)

[\\$explore_references\(\)](#)

[\\$explore_reference_parent\(\)](#)

[\\$goto_directory\(\)](#)

\$explore_parent()

Scope: dmgr_model

Prerequisite: To use this function, you must be in an active Navigator window.

Navigates up one level of the containment or reference network.

Usage

\$explore_parent()

EXPlore PArent

Navigator window > **Explore** > **Parent**

Arguments

None.

Return Values

VOID.

Description

The interactive function \$explore_parent() navigates up one level of the containment hierarchy or reference network and causes the navigator to display the contents of the selected design object or directory. In the case of references, the parent need not be a container.

You can press the ExploreParent key, as a shortcut method for executing the \$explore_parent() function, as noted in “[Key Names Mapped to Functions & Scopes](#)” on page 804.

Examples

1. This example explores the next higher level of containment or reference network.

```
$explore_parent();
```

2. This example explores the next higher level of the containment or reference network by using command syntax from the popup command line.

```
exp pa
```

Related Topics

[\\$explore_contents\(\)](#)

[\\$explore_references\(\)](#)

[\\$explore_reference_parent\(\)](#)

[\\$goto_directory\(\)](#)

[Specifying the Build Rules](#)

\$explore_reference_parent()

Scope: dmgr_model

Prerequisite: To use this function, you must be viewing the references of a selected design object in the Navigator window and you must have exactly one reference selected.

Navigates to and displays the contents of the parent directory of the referenced object.

Usage

\$explore_reference_parent()

EXPLore REference Parent

Arguments

None.

Return Values

VOID.

Description

The interactive function \$explore_reference_parent() navigates to and displays the contents of the parent directory of the referenced object. You can only execute this function when you are in *reference mode* and have a single reference selected.

You are in reference mode after you execute the \$explore_references() function. You can verify that you are in reference mode by viewing the navigator title bar; if you are in reference mode, the right-most character is an “@”. When you execute the \$explore_reference_parent() function, the contents of the referenced object's parent container are displayed in the Navigator window and you are returned to *contents mode*.

In reference mode, you can execute both the \$explore_parent() function and the \$explore_reference_parent() function. The difference between these two functions is that the \$explore_parent() function navigates to and displays the parent container of the design object that holds the references, whereas the \$explore_reference_parent() function navigates to and displays the parent container of the design object to which the reference points.

Examples

1. This example first explores the references of the configuration object *double_config*, and then explores the parent of the design object pointed to by the reference */net/berg/double/schematic/sheet2*. After this example is executed, the navigator's title bar displays the path */net/berg/double/schematic*, which is the parent of the reference's target object *sheet2*.

```
$select_object("$PROJ_X/double_config", "Dme_config_do",
@reselect); $explore_references();
$select_object("/net/berg/double/schematic/sheet2",
"mgc_sheet", @reselect); $explore_reference_parent();
```

2. This example first explores the references of the configuration object *double_config*, just as in the previous example. However, in the second step, this example explores the parent of the design object which holds the reference */net/berg/double/schematic/sheet2*. After this example is executed, the navigator's title bar displays the path *\$PROJ_X*, which is the parent of the design object that holds the reference */net/berg/double/schematic/sheet2*.

```
$select_object("$PROJ_X/double_config", "Dme_config_do",  
@reselect); $explore_references();  
$select_object("/net/berg/double/schematic/sheet2",  
"mgc_sheet", @reselect); $explore_parent();
```

Related Topics

[\\$explore_contents\(\)](#)

[\\$\\$get_object_references\(\)](#)

[\\$explore_parent\(\)](#)

[\\$goto_directory\(\)](#)

\$explore_references()

Scope: dmgr_model

Prerequisite: To use this function, you must be in an active Navigator window and must select exactly one design object.

Navigates down one level of the reference network.

Usage

\$explore_references()

EXPlore REferences

Navigator window > **Explore** > **References**

Arguments

None.

Return Values

VOID.

Description

The interactive function \$explore_references() navigates to the design objects that are referenced by the selected design object. This function displays the absolute pathname of each reference in the active Navigator window. Reference pathnames are displayed exactly as they are stored in the attribute file. If you select a design object with no references, the navigator displays an empty window.

When exploring a design object's references using this function, the navigator adds that object's name to the title bar and follows the name with a “@”, indicating reference navigation. After exploring a design object's references, you can select another object in the display and explore its references or contents.

When you alternate between exploring contents and references, the title bar reflects this by displaying both “/” and “@” in the pathname, signifying either a content or reference exploration.

Examples

1. This example explores references of the selected design object.

\$explore_references();

2. This example explores references of the selected design object by using command syntax from the popup command line.

exp re

Related Topics

[\\$explore_contents\(\)](#)

[\\$explore_parent\(\)](#)

[\\$\\$get_object_references\(\)](#)

[\\$goto_directory\(\)](#)

\$export_configuration_entries()

Scope: dm_config_area

Prerequisite: To use this function, you must be in an active Configuration window.

Writes an ASCII representation of the contents of a Configuration window.

Usage

`$export_configuration_entries("path_name", portion)`

EXPort COnfiguration Entries *path_name portion*

File > Export Configuration

Arguments

- **path_name**

A string that specifies the pathname of the ASCII file. You can specify a relative or absolute pathname. If you specify a relative pathname, the Pyxis Project Manager application assumes that the path is relative to the current working directory. If the file specified in this pathname already exists, this function overwrites the file.

- **portion**

A switch name that specifies whether to write out all of the entries in the Configuration window. Options include:

- **@all (-All)** — Writes out all entries. Default.
- **@selected (-Selected)** — Writes out only selected entries.

Return Values

VOID.

Examples

1. This example creates an ASCII file called */user/joe/config_file* that contains all of the entries in the currently active Configuration window.

```
$export_configuration_entries("/user/joe/config_file", @all);
```

Related Topics

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

[\\$open_configuration_window\(\)](#)

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

\$export_library()

Scope: dmgr_project_model

Prerequisite: To use this function, a library must be selected in the Project Navigator window.

Creates a copy of the specified project library as an external library enabling project data to be re-used within other projects.

Usage

```
$export_library("library_path", "dest_dir_path", "new_ext_lib_name");
```

File > Export > External Library

Arguments

- **library_path**
The path to the library to be exported.
- **dest_dir_path**
The path to the directory in which the new external library resides.
- **new_ext_lib_name**
The name of the new external library which is to be created.

Return Values

If the export operation is successful the return value is a string containing the path to the new external library, otherwise the return is VOID.

Description

The source project library is not modified during this operation. The new external library uses the source project's technology settings.

Examples

The following example creates a copy of the project library at */tmp/project/lib* at */tmp/ext_lib*:

```
$export_library("/tmp/project/lib", "/tmp", "ext_lib");
```

Related Topics

[\\$create_dm_ext_lib\(\)](#)

[\\$import_icstudio_library\(\)](#)

[\\$import_ext_lib\(\)](#)

\$export_location_map()

Scope: session_area

Prerequisite: You can use this function from any Pyxis Project Manager window.

Causes the current location map to be written to a temporary file.

Usage

\$export_location_map()

EXPort LOfation Map

MGC > Location Map > Export Map

Arguments

None.

Return Values

VOID.

Description

These temporary files are created in \$MGC_HOME/tmp/ and are named *lmap.<unique ID>* where ID is a combination of letters followed by the process ID of the location map exporter. You can remove the temp files when all processes using the exported maps have finished.

Examples

This example causes the current in-memory location map to be written to a file named *lmap.<current_map_name>* in \$MGC_HOME/tmp. The environment variable MGC_LOCATION_MAP is set to the path to this file, thus exporting the current location map to any tool invoked from the current session.

```
$export_location_map();
```

Related Topics

[\\$change_location_map_entry\(\)](#)

[\\$show_location_map\(\)](#)

\$find_external_deps()

Scope: dmgr_proj_tk

Reports all external library dependencies for the selected Design Objects.

Usage

```
$find_external_deps([selected_objects])
```

Arguments

- **selected_objects**

A vector of vectors of strings. Each vector of strings represents the path and type of an object that should be searched for references to external library objects.

Return Values

Returns a vector with two elements. The first element is a Boolean indicating success (@true) or failure (@false). If the first element is true, then the second element is a vector listing the paths and types of all objects that reference design objects in external libraries.

Examples

The following example reports all references to external library data from any design object in the project at */tmp/proj1*:

```
$find_external_deps([[ "/tmp/proj1", "mgc_project" ]]);
```

\$find_references()

Scope: dmgr_proj_tk

Finds and reports all design objects that reference a specified target design object.

Usage

`$find_references("target_path", "target_type", [search_path], inclusive)`

FINd REferences

Report > Object References

Arguments

- **target_path**
A string specifying the path to the target object.
- **target_type**
A string specifying the type of the target object.
- **search_path**
A vector of strings specifying hierarchies that are searched for references to the target object.
- ***inclusive***
A Boolean with a default value of @false. Specifying @true causes the search to include all references to objects contained within the target object.

Return Values

Returns a vector listing the paths and types of all objects that reference the target object.

Examples

The following example reports all references to the schematic at *\$PROJ/lib1/cell1/schematic* from objects within the *\$PROJ* project:

```
$find_references("$PROJ/lib1/cell1/schematic", "mgc_schematic", ["$PROJ"]);
```

Related Topics

[\\$find_external_deps\(\)](#)

[\\$list_references\(\)](#)

\$\$fix_relative_path()

Scope: dme_dn_tk

Generates an absolute pathname from a relative pathname.

Usage

```
$$fix_relative_path("path_name", "parent_name")
```

Arguments

- **path_name**
A string that is the relative pathname to be converted to an absolute pathname.
- **parent_name**
A string that is a prefix to be appended to the beginning of the path_name argument.

Return Values

The absolute fixed equivalent pathname of the specified path_name string, or "NULL".

Description

The toolkit function `$$fix_relative_path()` generates an absolute pathname for the specified relative pathname. If the path_name argument begins with a "/", the parent_name directory is ignored. If the parent_name directory is not specified, the MGC working directory is used. This function returns the full fixed pathname that results from the concatenation of the parent_name directory and the path_name argument, or the concatenation of the MGC working directory and the path_name argument.

Path_name arguments of the form "anything/" are converted to the form "anything". Path_name arguments of the form "/./" are converted to the form "/". Path_name arguments of the form "/anything/.." are converted to the form "/".

Examples

This function returns the full pathname for specified relative pathname *my_dir*.

```
$$fix_relative_path("my_dir");
```

Related Topics

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

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

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

[\\$\\$is_relative_path\(\)](#)

[\\$get_navigator_directory_hard\(\)](#)

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

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

\$\$freeze_configuration()

Scope: dm_config_tk

Freezes all design object versions in the active configuration.

Usage

\$\$freeze_configuration()

Arguments

None.

Return Values

VOID.

Description

The toolkit function \$\$freeze_configuration() freezes each design object version in the active configuration. Freezing each configuration entry protects it from the version depth mechanism, which deletes versions that are beyond the default version depth.

This function cannot freeze unversioned design objects.

Examples

This example opens the configuration *my_config* and freezes it.

```
$$open_configuration("$PROJECT_XYZ/d1/my_config",0, @read_write);
```

```
$$freeze_configuration();
```

Related Topics

[\\$\\$unfreeze_configuration\(\)](#)

\$freeze_configuration()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window.

Freezes all design object versions in the active Configuration window and creates a new version of the configuration.

Usage

\$freeze_configuration()

FRoze COnfiguration()

File > Freeze

Configuration window popup menu > **Global Operations** > **Freeze**

Arguments

None.

Return Values

VOID.

Description

The interactive function \$freeze_configuration() freezes all design object versions in the active Configuration window. Freezing a design object's versions protects them from being deleted once they are beyond the current version depth. If the freeze operation is successful, a new version of the configuration object is created and numbered sequentially.

Because unversioned design objects do not keep previous versions, this function does not freeze unversioned design objects if they are part of the configuration.

You can interrupt the execution of the \$freeze_configuration() function, at any time, by pressing the Kill key. When you halt the freeze operation, a dialog box appears prompting you to abort or to continue the operation. If you choose to abort, the operation is halted. Design objects that have been frozen prior to the interrupt remain frozen; all others remain unfrozen.

To unfreeze a frozen configuration, you use the \$unfreeze_configuration() function. To view the status of the configuration, you use the \$report_configuration_info() function. To delete a frozen configuration, you use the \$delete_configuration() function.

Examples

1. This example freezes the design object versions in an active Configuration window and creates a new version of the configuration object.

```
$freeze_configuration();
```

2. This example freezes the design object versions in the active Configuration window using command syntax from the popup command line.

fre co

Related Topics

[\\$freeze_version\(\)](#)

[\\$unfreeze_configuration\(\)](#)

[\\$report_configuration_info\(\)](#)

[\\$delete_configuration\(\)](#)

\$\$freeze_version()

Scope: dme_do_tk

Freezes a version of the specified design object.

Usage

`$$freeze_version("obj_name", "obj_type", version)`

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **obj_type**
A string that specifies the type of the design object.
- **version**
An integer that specifies the version number of the design object. The default is the current version.

Return Values

VOID.

Description

The toolkit function `$$freeze_version()` freezes the specified version of the specified design object.

Freezing a version protects it from deletion once the version is beyond the specified version depth. Frozen versions are also protected from the `$$revert_version()` and `$$delete_version()` functions. Frozen versions are *not* protected from the `$$delete_object()` function, which deletes the entire design object.

If you omit the version argument, this function freezes the current version of the specified design object. To freeze a previous version of the design object, you must specify the version argument.

Examples

This example freezes version 4 of the design object "assembly".

```
$$freeze_version("$PROJ_XYZ/d1/assembly", "Document_file", 4);
```

Related Topics

[\\$\\$delete_version\(\)](#)

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

[\\$\\$get_version_depth\(\)](#)

[\\$\\$revert_version\(\)](#)

[\\$\\$set_version_depth\(\)](#)

[\\$\\$unfreeze_version\(\)](#)

\$\$delete_object()

\$freeze_version()

Scope: dmgr_version_model

Prerequisite: To use this function, you must be in an active version window and must select at least one version.

Freezes the selected design object version.

Usage

\$freeze_version()

FReeze VErsion

Edit > Freeze

Versions window popup menu > **Freeze**

Arguments

None.

Return Values

VOID.

Description

The interactive function \$freeze_version() freezes the selected design object version. Freezing a design object version protects it from automatic deletion when it is beyond the design object's current version depth. The \$\$delete_version() function and the \$revert_version() function cannot delete a frozen version; however, the \$delete_object() function can delete a design object that has frozen versions.

To unfreeze a frozen version, you use the \$unfreeze_version() function.

Examples

1. This example freezes the selected design object version.

\$freeze_version();

2. This example freezes the selected design object version by using command syntax from the popup command line.

fre ve

Related Topics

[\\$freeze_configuration\(\)](#)

[\\$report_version_info\(\)](#)

[\\$\\$delete_version\(\)](#)

[\\$show_versions\(\)](#)

[\\$unfreeze_version\(\)](#)

[\\$revert_version\(\)](#)

\$delete_object()

\$get_area_selected_objects()

Scope: dmgr_navigator_window

Prerequisite: To use this function, you must be in an active Navigator window.

Returns the pathnames and types of the selected design objects in the Navigator window.

Usage

`$get_area_selected_objects(mode)`

Arguments

- **mode**

A name that specifies whether this function returns the absolute pathname or leaf name of each of the selected design objects.

Choose one of the following:

- **@leaf** — The leaf name of each of the selected design objects is returned. Default.
- **@full** — The absolute pathname of each of the selected design objects is returned.

Return Values

A vector of vectors. Each sub-vector contains a pathname and a type. If no objects are selected, this function returns the empty vector []. If an error occurs, this function returns UNDEFINED. The format of the vector of vectors is:

```
[["object1", "type1"], ["object2", "type2"], ..., ["objectN", "typeN"]]
```

The first element of each subvector is a string that specifies either the absolute or relative pathname of a selected object. The second element is a string that specifies the type of the selected object.

An example of a returned value:

```
[["$PROJ_X/design1/config4", "Dme_config_do"]]
```

Description

The interactive function `$get_area_selected_objects()` returns the pathname and type of each of the currently selected objects in the active navigator. This function displays the returned value in the transcript window.

If possible, this function returns a soft pathname for each of the selected objects. You can use the `$$get_hard_name()` function to find the hard pathname equivalent of any soft pathname. You can use the `$$get_soft_name()` function to find the soft pathname equivalent of any hard pathname.

Examples

This example returns the absolute pathname and type of each of the selected objects in the active navigator to the transcript window.

```
$get_area_selected_objects(@full);
```

Related Topics

[\\$\\$get_container_contents\(\)](#)

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

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

\$\$get_children()

Scope: dm_config_tk

Returns the children of a configuration entry.

Usage

```
$$get_children("name", "type", version)
```

Arguments

- **name**
A string that specifies the pathname of the configuration entry.
- **type**
A string that specifies the type of the configuration entry.
- ***version***
An integer that specifies the version number of the configuration entry. The default is 0, which represents the current version.

Return Values

A vector of vectors. Each sub-vector contains three elements: a name, a type, and a version number. If an error occurs, this function returns UNDEFINED.

Description

The toolkit function `$$get_children()` returns the children of the specified configuration entry. If the specified configuration entry is a container, this function returns all entries in the configuration that are contained by the container. This function does not return all design objects contained by the container in the file system, but only those design objects that are part of the configuration.

This function returns the children in a vector of string vectors. The first element of each subvector is a string that specifies the absolute pathname of the child. The second element is a string that specifies the type of the child. The third element is an integer that specifies the version number.

The format of the vector of string vectors is:

```
[["child1", "type1", version], ["child2", "type2", version], ..., ["childN", "typeN", version]]
```

Examples

The following example opens the configuration "my_config" and assigns the children of container "project" to a variable named "children".

```
$$open_configuration("$PROJECT_XYZ/design1/my_config", 0,  
    @read_write);  
  
local children = $$get_children("$PROJECT_XYZ/project",  
    "Mgc_container", 0);
```

The variable *children* now contains the following:

```
[["$PROJECT_XYZ/project/file", "Mgc_file", 0],  
["$PROJECT_XYZ/project/staff", "Dss_sheet", 2],  
["$PROJECT_XYZ/project/schedule", "Image_file", 4]]
```

Related Topics

[\\$\\$get primaries\(\)](#)

[\\$\\$is_entry_container\(\)](#)

[\\$\\$get_secondaries\(\)](#)

\$\$get_configuration_entries()

Scope: dm_config_tk

Returns all entries in the active configuration.

Usage

\$\$get_configuration_entries()

Arguments

None.

Return Values

A vector of string vectors. If an error occurs, this function returns UNDEFINED.

Description

The toolkit function \$\$get_configuration_entries() returns a list of entries in the active configuration.

The format of the vector of string vectors is:

```
[["object1", "type", version], ["object2", "type2", version], ["objectN", "typeN", version]]
```

The first element of each subvector is a string that specifies the absolute pathname of the configuration entry. The second element is a string that specifies the type of the configuration entry. The last element is an integer that specifies the version number of the configuration entry.

Examples

This example opens the configuration “config_galaxy” and assigns the entries to a variable called “entries”.

```
$$open_configuration("$PROJECT_XYZ/config_galaxy",0,@read_write);  
local entries = $$get_configuration_entries();
```

The variable entries now contains the following:

```
[["$PROJECT_XYZ/planets", "Mgc_container", 2],  
["$PROJECT_XYZ/planets/saturn", "Mgc_file", 4],  
["$PROJECT_ABC/moons", "Mgc_container", 1],  
["$PROJECT_ABC/moons/saturn", "Dss_sheet", 5],  
["$PROJECT_ABC/moons/earth", "Dss_sheet", 3],  
["$PROJECT_MNO/sun", "Mgc_container", 0],  
["$PROJECT_MNO/sun/spots", "Image_file", 2]]
```

Related Topics

[\\$\\$get_configuration_path\(\)](#)

[\\$\\$get_secondaries\(\)](#)

[\\$\\$get primaries\(\)](#)

\$\$get_configuration_path()

Scope: dm_config_tk

Returns the soft (absolute) pathname of the active configuration.

Usage

```
$$get_configuration_path()
```

Arguments

None.

Return Values

A string that specifies the absolute pathname of the configuration. If an error occurs, this function returns UNDEFINED.

Examples

This example opens the configuration object “config_x” and assigns the absolute pathname of the configuration to a variable called “path”.

```
$$open_configuration("$PROJECT_XYZ/d1/config_x", 0, @read_write);  
    local path = $$get_configuration_path();
```

Related Topics

[\\$\\$get_configuration_entries\(\)](#)

[\\$\\$get_secondaries\(\)](#)

[\\$\\$get primaries\(\)](#)

\$\$get_container_contents()

Scope: dme_do_tk

Returns the name, type, and version of the objects in the specified container.

Usage

```
$$get_container_contents("obj_name", "type")
```

Arguments

- **obj_name**
A string that specifies the name of the directory or container design object.
- **type**
A string that specifies the type of the design object. The default is "Mgc_container".

Return Values

A vector of string vectors. Each of the string vectors contains the name, type, and version number of a child of the specified container. If the container is empty or does not exist, this function returns VOID.

Description

The toolkit function `$$get_container_contents()` returns a vector that lists the design objects that exist in the specified container.

The format of the vector is:

```
[["object1", "type1", version], ["object2", "type2", version], ["objectN", "typeN", version]]
```

Examples

This example returns the contents of the container object *mgc*.

```
$$get_container_contents("$PROJECT_XYZ/stuff/mgc","Mgc_container");
```

The preceding example returns the following string vector:

```
[["file1", "Mgc_file", 1], ["example_config", "Dme_config_do", 3]]
```

Related Topics

[\\$\\$add_directory\(\)](#)

[\\$\\$is_directory\(\)](#)

[\\$\\$is_container\(\)](#)

\$\$get_date_last_modified()

Scope: dme_do_tk

Returns the date that the specified design object was last saved to disk.

Usage

```
$$get_date_last_modified("obj_name", "type")
```

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **type**
A string that specifies the type of the design object.

Return Values

A string that specifies the date on which the design object was last saved to disk. If the specified design object does not exist, this function returns <NULL VALUE>.

Examples

This example returns the date that the design object *assembly* was last modified.

```
$$get_date_last_modified("$PROJECT_XYZ/assembly", Mgc_file);
```

The preceding example returns the following:

```
Thu Nov 8 10:48:49 1990
```

Related Topics

[\\$\\$get_fileset_members\(\)](#)

[\\$\\$get_object_parent_path\(\)](#)

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

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

\$get_default_tool()

Scope: dmgr_tk

Returns the default tool for the specified type.

Usage

```
$get_default_tool("object_type")
```

Arguments

- **object_type**
A string that specifies the type of the design object.

Return Values

A string that specifies the name of the default tool.

This example returns the name of the default tool for the specified type and assigns the tool name to a variable named "default_tool".

```
local default_tool = $get_default_tool("mgc_sheet");
```

The variable "default_tool" in the preceding example now contains the following string:

```
"design_arch"
```

Description

The interactive function `$get_default_tool()` returns a string that specifies the default tool for the specified type. The default tool is the tool that operates on the design object when no other tool is specified. This function returns the value of the "default_tool" property, which is held by the specified type.

This function enables you to determine the default tool when more than one tool appears in the popup menu option **Open** from the Navigator window or from the version window.

\$\$get_entry_version()

Scope: dm_config_tk

Returns the version number of the target object of a current entry in the active configuration.

Usage

```
$$get_entry_version("name", "type")
```

Arguments

- **name**
A string that specifies the pathname of the design object whose version number is being returned.
- **type**
A string that specifies the type of the design object whose version number is being returned.

Return Values

An integer. If the specified configuration entry is not a current entry, an error is returned. If the specified configuration entry does not exist, this function returns UNDEFINED.

Description

The version number that is returned may or may not be the current version of the target object, depending on how the object has changed since the last configuration build.

Examples

This example returns the version number of the target object of the configuration entry *\$PROJECT_XYZ/project/config_docs*.

```
$$get_entry_version("$PROJECT_XYZ/project/config_docs", "mgc_sheet");
```

Related Topics

[\\$\\$is_configuration_edited\(\)](#)

[\\$\\$is_configuration_locked\(\)](#)

[\\$\\$is_configuration_frozen\(\)](#)

\$\$get_fileset_members()

Scope: dme_do_tk

Returns the fileset members of the specified design object.

Usage

`$$get_fileset_members("obj_name", "type", version)`

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **type**
A string that specifies the type of the design object.
- **version**
An integer that specifies the version number of the design object. The default is the current version.

Return Values

A vector of strings. Each string is a fileset member of the specified design object.

Description

Returns a vector of strings, which are fileset members of the specified design object. To get the fileset members of a previous version, you must specify the version argument.

This function does not return fileset members with the extensions *.frz*, *.ed*, *.lck*, *.ref*, *.ref_frz*, and *.attr*.

Examples

This example returns the fileset members of version 2 of the “vegetables” design object.

```
$$get_fileset_members("$PROJECT_XYZ/d1/vegetables", "Image_file", 2);
```

The preceding example returns the following vector:

```
[ "vegetables.img_2" ]
```

Related Topics

[\\$\\$get_date_last_modified\(\)](#)

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

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

[\\$\\$object_complete\(\)](#)

[\\$\\$get_object_parent_path\(\)](#)

[\\$\\$object_exists\(\)](#)

\$\$get_hard_name()

Scope: dme_dn_tk

Returns the hard pathname equivalent of any pathname.

Usage

```
$$get_hard_name("path_name")
```

Arguments

- **path_name**
A string that contains any pathname.

Return Values

A hard pathname.

Description

The toolkit function `$$get_hard_name()` returns the hard pathname equivalent of any soft pathname, based on the values found in the current in-memory location map and any existing environment variables.

If you enter a pathname for the `path_name` argument that begins with a dollar sign (\$), the first element of the path is assumed to be either a soft prefix in your location map or an environment variable. The first element of the path, the characters between the initial (\$) and the first slash (/), is extracted and the location map is searched for a matching soft prefix. If a match is found, the first hard pathname associated with the matching soft prefix is appended to the beginning of the remaining pathname string and this function returns the resulting path.

If the function can't find a matching soft prefix, the function initiates a search for a matching environment variable. If a matching environment variable is found, the value of the environment variable is appended to the beginning of the remaining pathname string and the function returns the resulting path. If a matching environment variable is not found, the function returns a null string and puts an error on the global status stack, which you can obtain with the `$$report_global_status()` or `$$get_global_status()` functions.

If you enter a hard pathname for the `path_name` argument, the hard pathname is returned. If you enter a relative pathname, this function resolves it by appending the value of the MGC working directory to the beginning of the relative pathname.

Examples

1. This example returns the hard pathname equivalent of the soft prefix `$PROJ_DESIGN`.

```
$$get_hard_name("$PROJ_DESIGN");
```
2. This example returns the hard pathname equivalent of the `my_dir` relative pathname .

```
$$get_hard_name("my_dir");
```

Related Topics

[\\$\\$fix_relative_path\(\)](#)

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

[\\$get_navigator_directory_hard\(\)](#)

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

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

[\\$\\$is_relative_path\(\)](#)

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

\$\$get_location_map()

Scope: dme_dn_tk

Returns the absolute path of the current location map, and the values of its entries.

Usage

```
$$get_location_map()
```

Arguments

None.

Return Values

A vector of string vectors that specifies the current location map and its entries:

```
[“location_map_file”, [“soft_prefix”, [“hard_name”,...,“hard_nameN”],  
  overridden_flag], [“soft_prefix2”, [“hard_name”,...,“hard_nameN”],  
  overridden_flag], [“soft_prefixN”, [“hard_name”,...,“hard_nameN”],  
  overridden_flag]]
```

The `location_map_file` is a string that specifies the absolute path of the current in-memory location map. Each `soft_prefix` is a string that specifies a soft prefix entry. The `hard_name` is a vector of strings that specifies zero to N physical paths that map to the soft prefix entry. `Overridden_flag` is an integer that specifies whether the soft prefix results from overriding the location map file.

Description

The toolkit function `$$get_location_map()` returns the absolute path of the current location map, and a listing of its entries and their sources.

The overridden integer returned may be one of two possible values:

- A value of 0, which indicates that the hard pathname results from the values in the location map.
- OR
- A value of 1, which indicates that the hard pathname results from either an environment variable or a call to the `$$set_location_map_entry()` function.

Examples

This example executes the `$$get_location_map()` function and lists the values of a sample output.

```
$$get_location_map();
```

Example of the returned vector:

```
["/usr/tmp/mgc_location_map",  
["$PAMJ", "//pamj/local_user/pamj", "/user/pamj", 0],  
["$DDMS_PROJ", "/project/ddms/src", 0],["$DMGR",  
"/project/dmgr/src/userware/default",  
"/dm/dmgr.proj/bin/dmgrv8.1_1.5/dme", 0]
```

Related Topics

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

[\\$\\$show_location_map\(\)](#)

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

\$\$get_monitor_error_count()

Scope: dm_config_tk

Returns the number of errors processed by the monitor since the monitor was last cleared.

Usage

`$$get_monitor_error_count()`

Arguments

None.

Return Values

An integer that indicates the number of errors that have been processed since the monitor was last cleared. Each time the monitor is cleared, the value of the integer is reset to 0.

Examples

This example returns the number of errors that have been processed by the monitor since the monitor was last cleared.

```
if ($$get_monitor_error_count() != 0) {  
    $message("Errors encountered during operation.");  
}
```

Related Topics

[\\$\\$get_monitor_verbosity\(\)](#)

[\\$\\$set_monitor_verbosity\(\)](#)

[\\$\\$set_monitor_flag\(\)](#)

[\\$setup_monitor\(\)](#)

\$\$get_monitor_flag()

Scope: dm_config_tk

Returns the value of the specified configuration monitoring attribute.

Usage

```
$$get_monitor_flag(index)
```

Arguments

- **index**

A name that specifies the configuration monitoring attribute whose value you want returned.

- **@noerrors** — Specifies whether error messages are filtered out of configuration monitor status reports. If you specify this attribute and the returned value is @true, error messages are not reported. If you specify this attribute and the returned value is @false, all error messages are reported.
- **@nowarnings** — Specifies whether warning messages are included in configuration monitor status reports. If you specify this attribute and the returned value is @true, warning messages are not reported. If you specify this attribute and the returned value is @false, all warning messages are reported.
- **@noinfo** — Specifies whether informational messages are included in configuration monitor status reports. If you specify this attribute and the returned value is @true, informational messages are not reported. If you specify this attribute and the returned value is @false, informational messages are reported.
- **@show_window** — Specifies whether the monitor window is automatically displayed during configuration operations. If you specify this attribute and the returned value is @true, the configuration monitor window is automatically displayed during configuration operations. If you specify this attribute and the returned value is @false, the configuration monitor window is not displayed during configuration operations.
- **@window_mode** — Specifies whether configuration monitoring status is output to the monitor window or to the transcript area. If you specify this attribute and the returned value is @true, configuration monitoring status is output to the monitor window only. If you specify this attribute and the returned value is @false, configuration monitoring status is output to the transcript area only.
- **@enable_clearing** — Specifies whether the configuration monitor window is automatically cleared upon each new configuration operation. If you specify this attribute and the returned value is @true, the configuration monitor window is cleared each time you execute a configuration operation. If you specify this attribute and the returned value is @false, the configuration monitor window is not cleared when you execute a new configuration operation. Instead, the most recent status

information is appended to the bottom of the information previously reported in the configuration monitor window.

Return Values

A Boolean that indicates the value of the attribute specified by the index argument.

Description

The toolkit function `$$get_monitor_flag()` returns the Boolean value of the configuration monitoring attribute that you specify in the index argument. You can use the `$$set_monitor_flag()` function to specify a new value for a configuration monitoring attribute.

Examples

This example returns the value of the `@noerrors` attribute, showing that no error messages are filtered from configuration status reporting.

```
$$get_monitor_flag(@noerrors);
```

```
// Returned value:
```

```
//@false
```

Related Topics

[\\$\\$get_monitor_verbosity\(\)](#)

[\\$\\$set_monitor_verbosity\(\)](#)

[\\$\\$set_monitor_flag\(\)](#)

[\\$setup_monitor\(\)](#)

\$\$get_monitor_verbosity()

Scope: dm_config_tk

Returns the value of the specified configuration monitoring attribute.

Usage

```
$$get_monitor_verbosity()
```

Arguments

None.

Return Values

An integer with a value in the range of 0 to 5. The integer corresponds to the verbosity setting of the configuration monitor, where a value of 1 is terse, a value of 5 is verbose, and all values in between are gradations of verbosity between the two extremes. A value of 0 indicates no ranking is applied; messages with a ranking of 0 are always printed regardless of the monitor verbosity setting.

Description

The toolkit function `$$get_monitor_verbosity()` returns the configuration monitor verbosity setting which is an integer ranging from 1 to 5. You can use the `$$set_monitor_verbosity()` function to specify a new value for the configuration monitoring verbosity.

Examples

This example returns the value of monitoring verbosity, showing that the setting is specified as the default value.

```
$$get_monitor_verbosity();
```

```
// Returned value:
```

```
// 2
```

Related Topics

[\\$\\$get_monitor_verbosity\(\)](#)

[\\$\\$set_monitor_verbosity\(\)](#)

[\\$\\$set_monitor_flag\(\)](#)

[\\$setup_monitor\(\)](#)

\$\$get_monitor_warning_count()

Scope: dm_config_tk

Returns the number of warnings processed by the monitor since the monitor was last cleared.

Usage

`$$get_monitor_warning_count()`

Arguments

None.

Return Values

An integer that indicates the number of warning messages that have been processed since the monitor was last cleared.

Examples

This example returns the number of warnings that have been processed by the monitor since the monitor was last cleared.

```
if ($$get_monitor_warning_count() != 0) {  
    $message("Warnings encountered during operation.");  
}
```

Related Topics

[\\$\\$get_monitor_verbosity\(\)](#)

[\\$\\$set_monitor_verbosity\(\)](#)

[\\$\\$set_monitor_flag\(\)](#)

[\\$setup_monitor\(\)](#)

\$get_navigator_directory()

Scope: dmgr_model

Prerequisite: To use this function, you must be in an active Navigator window.

Returns the soft pathname of the current navigator directory.

Usage

`$get_navigator_directory()`

Arguments

None.

Return Values

Returns the navigator's directory.

Example of a returned value:

```
"$PROJ_X/design2"
```

Examples

This example displays the soft pathname of the active navigator's current navigator directory in the transcript window.

The Navigator window uses the value returned by the `$get_navigator_directory()` function to resolve relative pathnames. All other Pyxis Project Manager windows use the value returned by the `$$get_working_directory()` function to resolve relative pathnames.

```
$get_navigator_directory();
```

Related Topics

[\\$get_navigator_directory_hard\(\)](#)

\$get_navigator_directory_hard()

Scope: dmgr_model

Prerequisite: To use this function, you must be in an active Navigator window.

Returns the hard pathname of the current navigator directory.

Usage

`$get_navigator_directory_hard()`

Arguments

None.

Return Values

The hard path of the navigator's working directory.

Sample returned value:

```
"//bradc/local_user/bradc/project/design2"
```

Description

The interactive function `$get_navigator_directory_hard()` returns the hard pathname of the active navigator's current navigator directory in the transcript window.

You can press the Navigator Directory key, as a short-cut method for executing the `$get_navigator_directory_hard()` function, as shown in “[Key Names Mapped to Functions & Scopes](#)” on page 804.

Examples

This example displays the hard path of the current navigator directory to the transcript window.

```
$get_navigator_directory_hard();
```

Related Topics

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

\$get_next_tool_env()

Scope: dm_tool_tk

Returns the value of the environment variable for the next tool.

Usage

```
$get_next_tool_env("name")
```

Arguments

- **name**
A string representing the name of the environment variable to query.

Return Values

The value of the given environment variable for the next tool. Void if the variable is not found.

Examples

```
$get_next_tool_env("Variable_A")  
"Value_A"
```

Related Topics

[\\$set_next_tool_env\(\)](#)

[\\$unset_next_tool_env\(\)](#)

\$\$get_object_current_version()

Scope: dme_do_tk

Returns the current version of the specified design object.

Usage

```
$$get_object_current_version("obj_name", "obj_type")
```

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **obj_type**
A string that specifies the type of the design object.

Return Values

An integer that specifies the current version of the specified design object.

Examples

This example returns the current version of the “section_4” design object.

```
$$get_object_current_version("$PROJECT_XYZ/d1/section_4", "Document_file");
```

Related Topics

[\\$\\$get_date_last_modified\(\)](#)

[\\$\\$get_object_parent_path\(\)](#)

[\\$\\$get_fileset_members\(\)](#)

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

\$\$get_object_parent_path()

Scope: dme_do_tk

Returns the absolute pathname of the parent of the specified design object. The parent is the directory or container that contains the specified design object.

Usage

```
$$get_object_parent_path("obj_name", "type")
```

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **type**
A string that specifies the type of the design object.

Return Values

A string that specifies the absolute pathname of the parent of the specified design object.

Examples

This example returns the parent of the “canvas” design object.

```
$$get_object_parent_path("$PROJECT_XYZ/example/canvas", Mgc_file);
```

The preceding example returns the following string:

```
"$PROJECT_XYZ/example"
```

Related Topics

[\\$\\$get_date_last_modified\(\)](#)

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

[\\$\\$get_fileset_members\(\)](#)

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

\$\$get_object_path_filter()

Scope: dm_config_tk

Returns the value of the object path filter for the specified primary entry.

Usage

```
$$get_object_path_filter("name", "type", version)
```

Arguments

- **name**
A string that specifies the pathname of the primary entry.
- **type**
A string that specifies the type of the primary entry.
- ***version***
An integer that specifies the version number of the configuration entry. The default is 0, which represents the current version.

Return Values

A vector of string vectors. If an error occurs, this function returns UNDEFINED.

Description

The object path filter either includes or excludes certain design objects from a configuration based on their pathname.

The format of the vector of string vectors is:

```
[[“path pattern1”, “path pattern2”, “path patternN”], [“include1”, “exclude2”, “includeN”]]
```

The path pattern is a string that is used to match against pathnames. You can use UNIX regular expressions to specify the path pattern. The include or exclude string determines if a design object is included or excluded from the configuration after a successful match is made.

To return the filter value of a previous version of a configuration entry, you must specify the version argument. Otherwise, this function returns the filter value of the current version.

To set the object path filter, you use the `$$set_object_path_filter()` function.

Examples

This example returns the object path filter value for the `$PROJECT_XYZ/project` configuration entry.

```
$$get_object_path_filter("$PROJECT_XYZ/project", "Mgc_container", 0);
```

The preceding example returns the following vector:

```
[ ["^$PROJECT_ABC/d1", "$PROJECT_ABC/d1/plans"], ["include", "exclude" ]]
```

This filter includes design objects that match the *\$PROJECT_XYZ/d1* pathname pattern in the configuration and excludes design objects that match the “*\$PROJECT_XYZ/d1/plans*” pathname pattern.

The regular expression (^) denotes that the pathname pattern must occur at the beginning of the string.

Related Topics

[\\$\\$clear_entry_filter\(\)](#)

[\\$\\$get_reference_property_filter\(\)](#)

[\\$\\$get_object_property_filter\(\)](#)

[\\$\\$set_object_path_filter\(\)](#)

[\\$\\$get_object_type_filter\(\)](#)

\$get_object_pathname()

Scope: dm_tool_tk

Prerequisite: This function is only available from within a qualification or termination script.

Returns the absolute pathname of the specified design object during data-centered tool invocation, or the absolute pathname of the active tool viewpoint during tool-centered tool invocation

Usage

`$get_object_pathname()`

Arguments

None.

Return Values

A string that specifies the absolute pathname of the specified design object during data-centered tool invocation, or the absolute pathname of the active tool viewpoint during tool-centered tool invocation.

Description

You can use this function in a qualification script to provide the absolute pathname of either the specified design object or the active tool viewpoint to the invoking tool.

Examples

This example assigns the selected design object's absolute pathname to a variable named "obj_path".

```
local obj_path = $get_object_pathname();
```

Related Topics

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

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

\$\$get_object_properties()

Scope: `dme_do_tk`

Returns the properties of the specified design object.

Usage

`$$get_object_properties("obj_name", "obj_type", version)`

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **obj_type**
A string that specifies the type of the design object.
- **version**
An integer that specifies the version number of the design object. The default is the current version.

Return Values

A vector of string vectors. Each vector contains the name and value of a property that is held by the specified design object. If the specified design object has no properties or does not exist, this function returns VOID.

Description

The toolkit function `$$get_object_properties()` returns a vector of string vectors that contains the properties that exist on the specified design object.

By default, this function returns the properties of the current version of the specified design object. To get the properties of a previous version of the design object, you must specify the version argument.

The format of the vector of string vectors is:

```
[[["property1", "value1"], ["property2", "value2"], ["propertyN", "valueN"]]
```

When you set an object property, it remains on the current version of the design object. The object property carries forward as the design object evolves and the version number is incremented. However, if you set an object property on a previous version, that property remains with that version and does not carry forward as the design object evolves.

Examples

This example returns the properties held by version 3 of the "amino_acids" design object.

```
$$get_object_properties("$PROJECT_XYZ/d1/amino_acids", "Image_file", 3);
```

The preceding example returns the following string vectors:

```
[[ "qa_status", "testing incomplete"], [ "project_status", "complete"],  
  [ "engineer", "john smith"], [ "test_number", "30"]]
```

Related Topics

[\\$\\$delete_object_property\(\)](#)

[\\$\\$has_object_property\(\)](#)

[\\$\\$get_object_property_value\(\)](#)

[\\$\\$set_object_property\(\)](#)

[\\$\\$get_version_properties\(\)](#)

\$\$get_object_property_filter()

Scope: dm_config_tk

Returns the value of the object property filter for the specified primary entry.

Usage

```
$$get_object_property_filter("name", "type", version)
```

Arguments

- **name**
A string that specifies the absolute pathname of the configuration entry.
- **type**
A string that specifies the type of the configuration entry.
- **version**
An integer that defines the version number of the design object. The default is 0, which represents the current version.

Return Values

A vector of string vectors. If an error occurs, this function returns UNDEFINED.

Description

The toolkit function `$$get_object_property_filter()` returns the value of the object property filter for the specified configuration entry. The object property filter either includes or excludes certain design objects from a configuration based on the properties that are held by the design object.

The format of the vector of string vectors is:

```
[["property_name1", "property_name2", "property_nameN"],  
 ["property_value1", "property_value2", "property_valueN"], ["include1",  
 "exclude2", "includeN"]]
```

Both the property name and property value are strings that define the property held by the design object. The include or exclude string determines whether a design object is included or excluded from the configuration.

To return the filter value of a previous version of a configuration entry, you must specify the version argument. Otherwise, this function returns the filter value of the current version.

To set the object property filter, you use the `$$set_object_property_filter()` function.

Examples

This example returns the object property filter for the `$PROJECT_XYZ/schedule` configuration entry.

```
$$get_object_property_filter("$PROJECT_XYZ/schedule", "Mgc_container", 0);
```

```
// The preceding example returns the following string vectors:
```

```
// [{"engineer", "release"}, ["Joe_Brown", "beta"], ["include", "exclude"]]
```

This filter includes design objects with the property name “engineer” and the value “Joe_Brown” in the configuration, and excludes design objects with the property name “release” and the value “beta”.

Related Topics

[\\$\\$clear_entry_filter\(\)](#)

[\\$\\$get_reference_property_filter\(\)](#)

[\\$\\$get_object_path_filter\(\)](#)

[\\$\\$set_object_property_filter\(\)](#)

[\\$\\$get_object_type_filter\(\)](#)

\$\$get_object_property_value()

Scope: dme_do_tk

Returns the specified property value of the specified design object.

Usage

\$\$get_object_property_value("obj_name", "obj_type", *version*, "prop_name")

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **obj_type**
A string that specifies the type of the design object.
- **prop_name**
A string that specifies the name of the property.
- **version**
An integer that specifies the version number of the design object. The default is the current version.

Return Values

A string that is the value of the object property.

Description

By default, this function returns the property value of the current version of the specified design object. To return the property value of a previous version of a design object, you must specify the version argument.

Examples

This example returns the value of the property "engineer" for version 3 of the design object "amino_acids".

```
$$get_object_property_value("$PRJ_XYZ/project/amino_acids",  
    "Image_file", 3, "engineer");
```

The preceding example returns the following string value:

```
"John Smith"
```

Related Topics

[\\$\\$delete_object_property\(\)](#)

[\\$\\$has_object_property\(\)](#)

[\\$\\$get_object_properties\(\)](#)

[\\$\\$set_object_property\(\)](#)

\$\$get_object_protection()

Scope: dme_do_tk

Returns the protection status of the specified design object.

Usage

```
$$get_object_protection("obj_name", "type")
```

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **type**
A string that specifies the type of the design object.

Return Values

A string of nine characters, that indicate the protection status of the specified design object. If the specified design object does not exist, this function returns VOID.

Description

The toolkit function `$$get_object_protection()` returns a string of nine characters, which indicate the protection status of the specified design object.

For a directory, this function interprets the permission to execute as permission to search the directory for a specified file. For design objects, this function returns the protection of the attribute file. If no attribute file exists, this function returns the protection of the key file. For file system objects (files and directories), this function returns the protection of the physical file system object.

This function uses UNIX System V protection syntax. UNIX specifies three types of users: the *user*, or owner of the object; the user's *group*; and *others*. The three permission settings are “r” for read, “w” for write, and “x” for execute. For example, an object with the following permission settings gives read, write, and execute permission to the user, the user's group, and all others.

```
“rwxrwxrwx”
```

In the following example, the user can read, write, and execute the specified file; the user's group can read and execute only; and all others cannot read, write, or execute the file.

```
“rwxr-x---”
```

Examples

The following example returns the protection status of the “canvas” design object.

```
$$get_object_protection("$PROJECT_XYZ/canvas", "Mgc_file");
```

Related Topics

[\\$\\$is_read_protected\(\)](#)

[\\$\\$is_write_protected\(\)](#)

[\\$\\$set_protection\(\)](#)

[\\$\\$set_protection_numeric\(\)](#)

\$\$get_object_references()

Scope: dme_do_tk

Returns the references of the specified design object.

Usage

`$$get_object_references("obj_name", "obj_type", version)`

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **obj_type**
A string that specifies the type of the design object.
- **version**
An integer that specifies the version number of the design object. The default is the current version.

Return Values

A vector of vectors. Each element of the vector lists the references of the specified source design object.

Description

The toolkit function `$$get_object_references()` returns a vector of vectors that lists the references of the specified source design object. By default, this function returns the references of the current version of the design object. To get the references of a previous version, you must specify the version argument.

The format of the vector of vectors is:

```
[[["target_object1", "target_type1", target_version, reference_state, handle,  
  target_object_uid1],  
  ["target_object2", "target_type2", target_version, reference_state, handle,  
  target_object_uid2],  
  ["target_objectN", "target_typeN", target_version, reference_state, handle,  
  target_object_uidN]]
```

The `reference_state` can be one of three values: `@current`, `@read_only`, or `@fixed`. The reference handle is an integer, which uniquely identifies the target design object.

Examples

This example returns the references of the "part" source design object.

```
$$get_object_references("$PROJECT/test_t18s6anslt/demos/sims/basic_dc_ramp/part",  
"Eddm_part", void, void);
```

The preceding example returns the following vectors:

```
[["$TEST_T18S6ANSLT/demos/sims/basic_dc_ramp/schematic", "mgc_schematic",  
1, @current, 0, "54411bb30bf7.03.7f.00.00.01.00.31.99"]]
```

Related Topics

[\\$\\$add_reference\(\)](#)

[\\$\\$delete_reference\(\)](#)

[\\$\\$change_object_references\(\)](#)

\$\$get_object_type()

Scope: dm_tool_tk

Returns the design object type of the specified design object.

Usage

```
$$get_object_type("obj_name")
```

Arguments

- **obj_name**

A string that specifies the pathname of the design object.

Return Values

A vector of strings. Each element of the vector is a string that specifies a design object type.

Description

Returns a vector of strings that specifies the type name of the design object. This vector usually contains only one type, but if more than one design object has the pathname specified by the `obj_name` argument, the function returns multiple strings.

Examples

This example returns the types of the “example” design object.

```
$$get_object_type("$PROJECT_XYZ/d1/project/example");
```

The preceding example returns the following vector:

```
[ "Mgc_container", "Dme_config_do" ]
```

Related Topics

[\\$\\$get_date_last_modified\(\)](#)

[\\$\\$get_object_parent_path\(\)](#)

[\\$\\$get_fileset_members\(\)](#)

[\\$\\$object_exists\(\)](#)

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

\$get_object_type()

Scope: dm_tool_tk

Prerequisite: This function is only available from within a qualification or termination script.

Returns the type of the specified design object during data-centered tool invocation, or the type of the active tool viewpoint during tool-centered tool invocation.

Usage

`$get_object_type()`

Arguments

None.

Return Values

A string that specifies the type of the specified design object during data-centered tool invocation, or the type of the active tool viewpoint during tool-centered tool invocation.

Examples

This example returns the type of the selected design object and assigns it to a variable named *obj_type*.

```
local obj_type = $get_object_type();
```

Related Topics

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

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

\$\$get_object_type_filter()

Scope: dm_config_tk

Returns the value of the object type filter for the specified primary entry.

Usage

`$$get_object_type_filter("name", "type", version)`

Arguments

- **name**
A string that specifies the pathname of the configuration entry.
- **type**
A string that specifies the type of the configuration entry.
- ***version***
An integer that specifies the version number of the configuration entry. The default is 0, which represents the current version.

Return Values

A vector of string vectors. If an error occurs, this function returns UNDEFINED.

Description

The toolkit function `$$get_object_type_filter()` returns the value of the object type filter for the specified configuration entry. The object type filter either includes or excludes certain design objects from the configuration based on the type of the design object.

The format of the vector of string vectors is:

```
[["type pattern1", "type pattern2", "type patternN"], ["include1", "exclude2", "includeN"]]
```

The type is a string that identifies the type of the design object. The include and exclude string determines whether a design object is included or excluded from the configuration.

To return the filter value of a previous version of a configuration entry, you must specify the version argument; otherwise, this function returns the filter value of the current version.

To set the object type filter, you use the `$$set_object_type_filter()` function.

Examples

This example returns the object type filter for the configuration entry `$PROJECT_XYZ/project/folder`.

```
$$get_object_type_filter("$PROJECT_XYZ/project/folder", "Mgc_container", 0);
```

The preceding example returns the following vectors:

```
[["^Mgc_file$", "^Mgc_container$"], ["exclude", "exclude"]]
```

This filter excludes design objects of type “Mgc_file” and “Mgc_container” from the configuration. Regular expressions are used to denote the beginning (^) and ending (\$) of the string.

Related Topics

[\\$\\$clear_entry_filter\(\)](#)

[\\$\\$get_object_path_filter\(\)](#)

[\\$\\$get_object_property_filter\(\)](#)

[\\$\\$get_reference_property_filter\(\)](#)

[\\$\\$set_object_type_filter\(\)](#)

\$get_object_version()

Scope: dm_tool_tk

Prerequisite: This function is only available from within a qualification or termination script.

Returns the version number of the specified design object during data-centered tool invocation, or the version number of the active tool viewpoint during tool-centered tool invocation.

Usage

`$get_object_version()`

Arguments

None.

Return Values

An integer that specifies the version number of the specified design object during data-centered tool invocation, or the version number of the active tool viewpoint during tool-centered tool invocation.

Description

In a qualification script, you can use this function to determine if the selected design object should be opened in edit or read-only mode. For example, if you open a previous version of a design object from the version window, your qualification script should open the design object in read-only mode.

Examples

This example returns the version number of the selected design object and assigns it to a variable named *obj_ver*.

```
local obj_ver = $get_object_version();
```

Related Topics

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

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

\$\$get_object_versions()

Scope: dme_do_tk

Returns the versions of the specified design object.

Usage

`$$get_object_versions("obj_name", "obj_type")`

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **obj_type**
A string that specifies the type of the design object.

Return Values

A vector of integers. Each element of the vector is an existing version of the specified design object.

Description

The toolkit function `$$get_object_versions()` returns a vector of integers that are existing versions of the specified design object.

Examples

This example returns the existing versions of the design object *sheet*.

```
$$get_object_versions("$PROJECT_XYZ/design/sheet", "Mgc_file");
```

The preceding example returns the following vector of integers:

```
[3, 4, 5, 6]
```

Related Topics

[\\$\\$delete_version\(\)](#)

[\\$\\$revert_version\(\)](#)

[\\$\\$freeze_version\(\)](#)

[\\$\\$set_version_depth\(\)](#)

[\\$\\$get_version_depth\(\)](#)

[\\$\\$unfreeze_version\(\)](#)

\$\$get_parent_entry()

Scope: dm_config_tk

Returns the parent of the specified configuration entry.

Usage

`$$get_parent_entry("name", "type", version)`

Arguments

- **name**
A string that specifies the pathname of the configuration entry.
- **type**
A string that specifies the type of the configuration entry.
- **version**
An integer that specifies the version number of the configuration entry. The default is 0, which represents the current version.

Return Values

A vector of strings. If an error occurs, this function returns UNDEFINED.

Description

The toolkit function `$$get_parent_entry()` returns the absolute pathname of the parent of the specified configuration entry. This function also returns the parent's type and version number.

The format of the vector of strings is:

```
[[ "parent pathname", "type", version ]]
```

The parent pathname is a string that specifies the absolute pathname of the parent of the configuration entry. The type is a string that specifies the type of the parent. The version is an integer that specifies the version number of the parent.

Examples

This example opens the configuration *config_b* and assigns the parent of the specified configuration entry to a variable named *parent*.

```
$$open_configuration("$PROJECT_XYZ/config_b");  
local parent = $$get_parent_entry("$PROJECT_DESIGN/d1/file",  
    "Document_file", 3);
```

The variable *parent* now contains the following:

```
[ "$PROJECT_DESIGN/d1", "Mgc_container", 0 ]
```

Related Topics

[\\$\\$get_children\(\)](#)

[\\$\\$is_entry_container\(\)](#)

[\\$\\$is_entry_fixed\(\)](#)

[\\$\\$is_entry_primary\(\)](#)

[\\$\\$is_entry_retargetable\(\)](#)

\$\$get primaries()

Scope: dm_config_tk

Returns the primary entries in the active configuration.

Usage

\$\$get primaries()

Arguments

None.

Return Values

A vector of string vectors. If an error occurs, this function returns UNDEFINED.

Description

The toolkit function \$\$get primaries() returns a list of the primary entries in the active configuration. A primary entry is a design object that you add to the configuration by using the \$\$add_configuration_entry() function.

This function returns the primary entries in a vector of string vectors. The format of the vector of string vectors is:

```
[["primary1", "type1", version], ["primary2", "type2", version],  
 ["primaryN", "typeN", version]]
```

Primary is a string that specifies the absolute pathname of the primary entry. The type is a string that specifies the type of the primary entry. Version is an integer that specifies the version number of the primary entry.

Examples

This example opens the configuration *config_parts* and assigns the primary entries to a variable named *primary_entries*.

```
$$open_configuration("$PROJECT_XYZ/project/config_parts", 0,  
    @read_write);
```

```
local primary_entries = $$get primaries();
```

Related Topics

[\\$\\$get_configuration_entries\(\)](#)

[\\$\\$get_secondaries\(\)](#)

[\\$\\$get_configuration_path\(\)](#)

[\\$\\$add_configuration_entry\(\)](#)

\$\$get_reference_properties()

Scope: dme_do_tk

Returns the properties of the specified reference.

Usage

```
$$get_reference_properties("obj_name", "obj_type", obj_version, "target_obj_name",  
    "target_type", target_version)
```

Arguments

- **obj_name**
A string that specifies the pathname of the source design object.
- **obj_type**
A string that specifies the type of the source design object.
- ***obj_version***
An integer that specifies the version number of the source design object. The default is the current version.
- **target_obj_name**
A string that specifies the pathname of the target design object.
- **target_type**
A string that specifies the type of the target design object.
- ***target_version***
An integer that specifies the version number of the target design object. The *target_version* also represents the state of the reference. If the *target_version* is greater than 0, the state of the reference this function operates on is "fixed". If omitted or specified as 0, the state of the reference this function operates on is "current". The default is the current version.

Return Values

A vector of string vectors. Each element of the vector contains the name and value of a property that is held by the specified reference. If the reference has no properties or does not exist, this function returns VOID.


Description

The `$$get_reference_properties()` toolkit function returns a vector of vectors that contains the properties that exist on the specified reference. To return the reference properties of a previous version of the source design object, you must specify the *object_version* argument.

Because other tools can create duplicate references, you can use the `$$get_reference_properties_handle()` function for more accuracy when returning reference properties.

If you omit the `target_version` argument, this function returns the reference properties of the *first* current reference whose name and type match the specified arguments. As the `target_version` argument is not specified, the state of the reference is current and the version number is not checked.

If you specify the `target_version` argument, this function returns the reference properties of the *first* fixed reference whose name, type, and version number match the specified arguments. As the `target_version` argument is specified, the state of the reference is fixed and the version number is checked.

 **Note** If duplicate references exist, the `$$get_reference_properties()` function returns the properties of the first matching reference and a warning status, which states that more than one reference was found and the properties returned may not be the properties that you intend to work with.

The format of the vector of vectors is:

```
[[“property1”, “value1”], [“property2”, “value2”], [“propertyN”, “valueN”]]
```

The property string contains the name of the property. The value string contains the value contained in the property.

Examples

This example returns the reference properties from the current version of the “amino_acids” source design object to version 2 of the “plan” target design object. As the `target_version` is specified, the reference is fixed.

```
$$get_reference_properties("$PROJECT_XYZ/amino_acids", "Image_file",  
1, "$PROJECT_XYZ/project/plan", "Document_file", 2);
```

The preceding example returns the following vectors:

```
[["engineer", "john"], ["status", "incomplete"], ["test_no", "3"]]
```

Related Topics

[\\$\\$delete_reference_property\(\)](#)

[\\$\\$has_reference_property\(\)](#)

[\\$\\$get_reference_properties_handle\(\)](#)

[\\$\\$set_reference_property\(\)](#)

\$\$get_reference_properties_handle()

Scope: `dme_do_tk`

Returns the reference properties of the specified design object by using the target reference handle.

Usage

```
$$get_reference_properties_handle("obj_name", "obj_type", obj_version, target_handle)
```

Arguments

- **obj_name**
A string that specifies the absolute pathname of the source design object.
- **obj_type**
A string that specifies the type of the source design object.
- **obj_version**
An integer that specifies the version number of the source design object. The default is the current version.
- **target_handle**
An integer that specifies the handle of the target design object.

Return Values

A vector of string vectors. Each vector contains the reference property name and value.

Description

Returning reference properties using the target handle enables you to specify distinctly which reference properties to return. To find the reference's target handle, you use the `$$get_object_references()` function.

To return the reference properties of a previous version of the source design object, you must specify the `object_version` argument.

The format of the vector of string vectors is:

```
[["property1", "value1"], ["property2", "value2"], ["propertyN", "valueN"]]
```

The property string contains the name of the property. The value string contains the value contained in the property.

Examples

The following example returns the reference properties of version 2 of the design object *plan*, using the target handle 7.

```
$$get_reference_properties_handle("$PROJECT_XYZ/plan",  
    "Document_file", 2, 7);
```


In this example, `$$get_reference_properties_handle()` returns the following:

```
[[{"writer", "mary"}, {"engineer", "john"}, {"creating_tool", "design  
manager"},  
{"status", "incomplete"}]]
```

Related Topics

[\\$\\$delete_reference_property\(\)](#)

[\\$\\$has_reference_property\(\)](#)

[\\$\\$get_reference_properties\(\)](#)

[\\$\\$set_reference_property\(\)](#)

[\\$\\$get_object_references\(\)](#)

\$\$get_reference_property_filter()

Scope: dm_config_tk

Returns the value of the reference property filter for the specified primary entry.

Usage

`$$get_reference_property_filter("name", "type", version)`

Arguments

- **name**
A string that specifies the pathname of the configuration entry.
- **type**
A string that specifies the type of the configuration entry.
- **version**
An integer that specifies the version number of the configuration entry. The default is 0, which represents the current version.

Return Values

A vector of strings. If an error occurs, this function returns UNDEFINED.

Description

The toolkit function `$$get_reference_property_filter()` returns the value of the reference property filter for the specified configuration entry. The reference property filter either includes or excludes design objects from a configuration based on the properties that the reference holds.

The format of the vector of string vectors is:

```
[["property1", "property2", "propertyN"], ["value1", "value2", "valueN"],  
 ["include1", "exclude2", "includeN"]]
```

Both the property name and property value are strings that define the property held by the reference. The include and exclude strings determine whether a design object is included or excluded from the configuration.

To return the filter value of a previous version of a configuration entry, you must specify the version argument; otherwise, this function returns the filter value of the current version.

To set the reference property filter, you use the `$$set_reference_property_filter()` function.

Examples

This example returns the value of the reference property filter for the current version of the configuration entry *project*.

```
$$get_reference_property_filter("$PROJECT_XYZ/d1/project",  
 "Mgc_container", 0);
```

The return value of this example is:

```
[["TestNo."], ["3A"], ["exclude"]]
```

This filter excludes design objects with the reference property “TestNo.” and the value “3A” from the configuration.

Related Topics

[\\$\\$clear_entry_filter\(\)](#)

[\\$\\$get_object_type_filter\(\)](#)

[\\$\\$get_object_path_filter\(\)](#)

[\\$\\$set_reference_property_filter\(\)](#)

[\\$\\$get_object_property_filter\(\)](#)

\$\$get_reference_traversal()

Scope: dm_config_tk

Returns the reference traversal setting of the specified primary entry.

Usage

`$$get_reference_traversal("name", "type", version)`

Arguments

- **name**
A string that specifies the pathname of the primary entry.
- **type**
A string that specifies the type of the primary entry.
- ***version***
An integer that specifies the version number of the primary entry. The default is 0, which represents the current version.

Return Values

An integer. If an error occurs, this function returns UNDEFINED.

Description

During a build, the Pyxis Project Manager application traverses design object references, if reference traversal is *on*. If the reference traversal is set to *on*, this function returns -1. Otherwise, the reference traversal is "off", and this function returns 0.

To set the reference traversal of a primary entry, you use the `$$set_reference_traversal()` function.

Examples

This example opens the configuration *config_beta* and assigns the reference traversal of the primary entry *project* to a variable named *rf_t*.

```
$$open_configuration("$PROJECT_XYZ/config_beta", 0, @read_write);  
  
local rf_t = $$get_reference_traversal("$PROJECT_XYZ/project",  
    "Image_file", 4);
```

Related Topics

[\\$\\$build_configuration\(\)](#)

[\\$\\$is_entry_primary\(\)](#)

[\\$\\$is_entry_fixed\(\)](#)

[\\$\\$set_reference_traversal\(\)](#)

\$\$get_secondaries()

Scope: dm_config_tk

Returns a list of the secondary entries of the specified primary entry.

Usage

```
$$get_secondaries("name", "type", version)
```

Arguments

- **name**
A string that specifies the pathname of the primary configuration entry.
- **type**
A string that specifies the type of the configuration entry.
- **version**
An integer that specifies the version number of the configuration entry. The default is 0, which represents the current version.

Return Values

A vector of strings. If an error occurs, this function returns UNDEFINED.

Description

The Pyxis Project Manager application adds secondary entries to a configuration when you execute the \$\$build_configuration() function, based on the build rules you specify for each primary entry.

This function returns the secondary entries in a vector of string vectors. The format of the vector of string vectors is:

```
[[“secondary1”, “type1”, version], [“secondary2”, “type2”, version],  
 [“secondaryN”, “typeN”, version]]
```

Secondary is a string that specifies the absolute pathname of the secondary entry. The type is a string that specifies the type of the secondary entry. The version is an integer that specifies the version number of the secondary entry.

Examples

This example opens the configuration “config_alpha” and assigns the primary entries to a variable named “secondaries”.

```
$$open_configuration("$PROJECT_XYZ/project/config_alpha", 0,  
    @read_write);  
  
local secondaries = $$get_secondaries("$PROJECT_XYZ/d1/test",  
    "Mgc_file", 0);
```

Related Topics

[\\$\\$get_configuration_entries\(\)](#)

[\\$\\$get_configuration_path\(\)](#)

[\\$\\$get primaries\(\)](#)

[\\$\\$build_configuration\(\)](#)

\$\$get_soft_name()

Scope: dme_dn_tk

Returns the soft pathname equivalent of the specified pathname, based on the entries in the current location map.

Usage

```
$$get_soft_name("path_name")
```

Arguments

- **path_name**

A string that specifies the pathname whose soft path equivalent is returned.

Return Values

A soft pathname.

Description

If you specify a pathname that does not begin with a dollar sign (\$) or a slash (/), it is assumed to be a relative pathname and the MGC working directory is appended to the beginning of the relative pathname before the pathname is resolved to its soft equivalent.

When you specify a pathname that begins with a dollar sign (\$), the design data management system (DDMS) checks to see if you are using a location map. If so, the first element of the pathname is assumed to be a soft prefix. If the first element of the pathname matches a soft prefix in the location map, the pathname is accepted. If the first element does not match a soft prefix in the location map, an error message is displayed. If you are not using a location map, the first element of the pathname is assumed to be an environment variable and it is accepted.

If the specified pathname begins with a slash (/), the location map is searched for a matching hard pathname entry. For a successful match to occur, the location map's hard pathname entry must have exactly the same characters as the specified pathname, and the character following the matched string in the specified pathname, if there is one, must be a slash (/). For example, the pathname */abc/def/hij* would match the location map hard pathname */abc/def*, but not the hard pathname */abc/de*. If a match is found, the associated soft prefix for that hard pathname entry is substituted in place of the matching portion and the resulting pathname is returned. If a match is not found, the unconverted hard pathname is returned.

The first line of the location map file is always the header and consists of the "MGC_LOCATION_MAP_*" string, where the asterisk identifies the version number of the map file format. This string may be followed by the "force" string. The strings must be separated by one or more spaces.

The design data management system (DDMS) reads this line to determine the appropriate file format version of the location map. The optional "force" string tells DDMS that when an attempt to convert a hard pathname to a soft pathname is not successful, DDMS should issue an error, rather than storing the hard pathname as a reference.

Examples

This example returns the soft path equivalent of the *//ddms/local_user/dm.proj* hard pathname.

```
$$get_soft_name("//ddms/local_user/dm.proj");
```

Related Topics

[\\$\\$fix_relative_path\(\)](#)

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

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

[\\$\\$is_relative_path\(\)](#)

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

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

[\\$get_navigator_directory_hard\(\)](#)

\$\$get_status_code()

Scope: dme_dn_tk

Returns the error message code at the top of the status stack.

Usage

\$\$get_status_code()

Arguments

None.

Return Values

An integer that corresponds to the error message code at the top of the status stack.

If this function returns a code of zero, the previous toolkit function call was successful. If the returned code is non-zero, an error occurred, and the previous toolkit function call was not successful.

Examples

This example checks if \$\$move_object() was successfully called.

```
function $move_object(destination : string)
{
    $$clear_global_status();
    local object_vector = $get_selected_objects(@full);
    overwrite = @cancel;
    $$move_object(object_vector, destination, overwrite,      @nocreate);
    if ($$get_status_code() != 0) {
        $$report_global_status();
        return;
    }
}
```

Related Topics

[\\$\\$clear_global_status\(\)](#)

[\\$\\$get_status_messages\(\)](#)

[\\$\\$get_status_code_stack\(\)](#)

[\\$\\$report_global_status\(\)](#)

\$\$get_status_code_stack()

Scope: dme_dn_tk

Returns all of the error message codes on the status code stack.

Usage

```
$$get_status_code_stack()
```

Arguments

None.

Return Values

A vector of integers. Each integer is a code that represents a status message in the status code stack.

Examples

This example returns the status code stack.

```
$$get_status_code_stack();
```

Description

Returns the entire contents of the status code stack. This function enables you to perform a particular action based on a value found in the status code stack. Using this function, you can evaluate all status codes in the status code stack. To get the message strings corresponding to the status codes, use the `$$get_status_messages()` function.

Related Topics

[\\$\\$clear_global_status\(\)](#)

[\\$\\$get_status_messages\(\)](#)

[\\$\\$get_status_code\(\)](#)

[\\$\\$report_global_status\(\)](#)

\$\$get_status_messages()

Scope: dme_dn_tk

Returns the error messages on the status stack.

Usage

\$\$get_status_messages()

Arguments

None.

Return Values

A vector of strings. Each string corresponds to the messages on the status stack for the previously executed function.

Description

Returns a vector of strings based on the error codes in the global status stack. You can use this function to search for status messages or to write the messages to a file.

Examples

This example returns the status messages.

```
$$get_status_messages();
```

Related Topics

[\\$\\$clear_global_status\(\)](#)

[\\$\\$get_status_code_stack\(\)](#)

[\\$\\$get_status_code\(\)](#)

[\\$\\$report_global_status\(\)](#)

\$get_subinvoke_mode()

Scope: dm_tool_tk

Returns the current tool invocation method.

Usage

`$get_subinvoke_mode()`

Arguments

None.

Return Values

Returns @xterm when transcript messages are being sent to a new xterm window.

Returns @file for transcript messages to a log file.

Examples

```
$get_sub_invoke_mode();
```

Related Topics

[\\$set_subinvoke_mode\(\)](#)

\$\$get_target_path()

Scope: dm_config_tk

Returns the target path of the specified primary configuration entry.

Usage

```
$$get_target_path("name", "type", version)
```

Arguments

- **name**
A string that specifies the pathname of the primary configuration entry.
- **type**
A string that specifies the type of the primary configuration entry.
- **version**
An integer that specifies the version number of the primary configuration entry. The default is 0, which represents the current version.

Return Values

A string. If an error occurs, this function returns UNDEFINED.

Description

The target path is a pathname that you can specify for a retargetable configuration entry. When you perform either a release or a copy of the configuration, the retargetable entry is released or copied to its target path, instead of to the target directory of the release or copy operation.

To set the target path of an entry, you use the \$\$set_target_path() function. To check if an entry is retargetable, you use the \$\$is_entry_retargetable() function.

Examples

This example opens the “my_config” configuration and assigns the target path of the “base” primary entry to a variable named “target”.

```
$$open_configuration("$PROJECT_XYZ/design1/my_config", 0,  
    @read_write);
```

```
local target = $$get_target_path("$PROJECT_XYZ/design2/base",  
    "Mgc_file", 0);
```

Related Topics

[\\$\\$is_entry_retargetable\(\)](#)

[\\$\\$set_target_path\(\)](#)

\$get_technology()

Scope: dm_proj_tk

Retrieves the path to the technology configuration currently in use by the specified project or external library.

Usage

```
$get_technology("root_hdo_path");
```

Arguments

- **root_hdo_path**

A string value containing the path to the project or external library for which the technology setting is requested.

Return Values

Returns a string containing the path to the technology configuration in use by the specified root hierarchical design object.

If the specified hierarchy is not using a technology configuration, as with the MGC standard cell libraries, an empty string is returned.

Examples

The following example retrieves the technology configuration in use by the external library at */tmp/ext_lib*:

```
local config_path = $get_technology("/tmp/ext_lib");
```

Related Topics

[\\$create_dm_tech_lib\(\)](#)

[\\$set_technology\(\)](#)

[\\$create_tech_config_object\(\)](#)

\$get_toolbox_search_path()

Scope: dmgr_tk

Prerequisite: You can use this function from any Pyxis Project Manager window.

Returns the current toolbox search path.

Usage

`$get_toolbox_search_path()`

Arguments

None.

Return Values

Returns a vector of one or more strings, which represents the current toolbox search path. Each element of the vector is an absolute pathname of a toolbox in the current toolbox search path.

Description

The toolbox search path specifies the order in which the Pyxis Project Manager application searches for tools. Each string in the vector is an absolute pathname of a toolbox that contains tool viewpoints.

Examples

This example returns the current toolbox search path to the message area of the session window by using command syntax from the popup command line.

```
$message($get_toolbox_search_path());
```

The following is an example of the return value of a toolbox search path.

```
[ "/user/test/dme_test/support/testbox1",  
  "/user/test/dme_test/support/testbox2",  
  "$MGC_HOME/toolbox" ]
```

You can also view the return value in a session transcript window by typing `$get_toolbox_search_path()` on the popup command line.

Related Topics

[\\$add_toolbox\(\)](#)

[\\$remove_toolbox\(\)](#)

[\\$save_toolbox_search_path\(\)](#)

[\\$set_toolbox_search_path\(\)](#)

[\\$view_toolboxes\(\)](#)

[\\$view_tools\(\)](#)

\$get_tool_pathname()

Scope: dm_tool_tk

Prerequisite: This function is only available from within a qualification or termination script.

Returns the absolute pathname of the active tool viewpoint during data-centered or tool-centered tool invocation.

Usage

`$get_tool_pathname()`

Arguments

None.

Return Values

Returns a string that specifies the absolute pathname of the active tool viewpoint during data-centered or tool-centered tool invocation.

Description

You can use this function in a qualification script with design object toolkit functions, to gather information about the active tool viewpoint.

Examples

Suppose that your tool viewpoint has a property called “tool_version”, which contains the version number of your tool. To get the value of the tool_version property, you could use the following code fragment:

```
local tool_version=$$get_object_property_value($get_tool_pathname(),
    $get_tool_type(),"tool_version");
```

Related Topics

[\\$get_tool_type\(\)](#)

\$get_tool_script()

Scope: dm_tool_tk

Prerequisite: This function is only available from within a qualification or termination script.

Returns the absolute pathname of the active tool viewpoint's qualification or termination script.

Usage

```
$get_tool_script("tool_property_value")
```

Arguments

- **tool_property_value**

A string that specifies whether to return the pathname of the qualification script or the pathname of the termination script.

Return Values

Returns a string that specifies the absolute pathname of the tool viewpoint's scripts. If the scripts are not found, this function returns VOID.

Description

To return the pathname of the qualification script, you specify “qual” for the tool_property_value argument. To return the pathname of the termination script, you specify “term” for the tool_property_value argument.

To find these scripts, this function looks in two places.

1. The function first checks the references held by the active tool viewpoint for a property named “tool_reference”, with a value equal to that specified by the tool_property_value argument (either qual or term). If the reference property exists, then this function traverses the reference and returns the pathname of either the qualification script or the termination script.
2. If the reference property “tool_reference” is not found, this function returns the pathname of the file that is contained by the tool viewpoint with the filename suffix that matches the tool_property_value argument. For example, if the tool_property_value argument is qual, this function returns the pathname of the file in the active tool viewpoint container that has the suffix *.qual*.

Examples

1. This example returns the pathname of the qualification script and assigns it to a variable named “qual_path”.

```
local qual_path = $get_tool_script("qual");
```
2. This example returns the pathname of the termination script and assigns it to a variable named “term_path”.

```
local term_path = $get_tool_script("term");
```

Related Topics

[\\$invoke_bgd_tool\(\)](#)

[\\$invoke_tool\(\)](#)

\$get_tool_type()

Scope: dm_tool_tk

Prerequisite: This function is only available from within a qualification or termination script.

Returns the type of the active tool viewpoint during data-centered or tool-centered tool invocation.

Usage

`$get_tool_type()`

Arguments

None.

Return Values

A string that specifies the type of the active tool viewpoint.

Description

You can use this function in your qualification scripts with design object toolkit functions, to gather information about the active tool viewpoint.

Examples

Suppose that your tool viewpoint has a property called “tool_version”, which contains the version number of your tool. To get the value of the tool_version property, you could use the following code fragment:

```
local tool_version=$get_object_property_value($get_tool_pathname(),$get_tool_type(),  
"tool_version");
```

Related Topics

[\\$get_tool_pathname\(\)](#)

[\\$invoke_tool\(\)](#)

[\\$invoke_bgd_tool\(\)](#)

\$\$get_type_properties()

Returns all the properties of the specified design object type.

Usage

```
$$get_type_properties("do_type")
```

Arguments

- **do_type**
A string that is the design object type.

Return Values

Returns a vector of string vectors that contains the properties that exist on the specified design object.

Examples

```
$$get_type_properties("Mgc_container");
```

The preceding example returns the following:

```
[[ "file_mgr_type", "Ddms_do_wip_fm"], [ "fileset_def", [ "!r!c" ] ],  
[ "large_icon",  
  [ "$MGC_HOME/registry/fonts/dm_object.icons", "F" ] ], [ "small_icon",  
  [ "$MGC_HOME/registry/fonts/dm_text.icons", "F" ] ]]
```

Related Topics

[\\$\\$get_type_property_value\(\)](#)

\$\$get_type_property_value()

Returns a property value of the specified design object type.

Usage

`$$get_type_property_value("do_type", prop_name)`

Arguments

- **do_type**
A string of the design object type.
- **prop_name**
The name of the object property.

Return Values

Returns the value of the specified property type.

Examples

```
$$get_type_property_value("mgc_lm", "large_icon");
```

The preceding example returns the following:

```
["$MGC_HOME/registry/fonts/simv_object.icons", "L"]
```

Related Topics

[\\$\\$get_type_properties\(\)](#)

\$\$get_version_depth()

Scope: dme_do_tk

Returns the version depth of the specified design object.

Usage

```
$$get_version_depth("obj_name", "obj_type")
```

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **obj_type**
A string that specifies the type of the design object.

Return Values

An integer that specifies the version depth of the specified design object.

Description

Version depth is specified by the type of the design object. A version depth of -1 enables the design object to have an unlimited number of previous versions.

Examples

This example returns the version depth of the design object *list*.

```
$$get_version_depth("$PROJECT_XYZ/project/list", "Document_file");
```

Related Topics

[\\$\\$delete_version\(\)](#)

[\\$\\$revert_version\(\)](#)

[\\$\\$freeze_version\(\)](#)

[\\$\\$set_version_depth\(\)](#)

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

[\\$\\$unfreeze_version\(\)](#)

[\\$\\$get_version_properties\(\)](#)

\$\$get_version_properties()

Scope: dme_do_tk

Returns the properties of the specified design object version.

Usage

```
$$get_version_properties("obj_name", "obj_type", version)
```

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **obj_type**
A string that specifies the type of the design object.
- **version**
An integer that specifies the version number of the design object. The default is the current version.

Return Values

A vector of string vectors that contains the properties existing on the specified design object version. Each element of the vector contains a version property that is held by the specified design object version. If the design object version has no properties or does not exist, this function returns VOID.

Description

Version properties always remain with the version to which they were originally set. As the design object evolves, a version property does not carry forward to the next version.

To return the version properties of a previous version of the design object, you must specify the version argument; otherwise, this function returns the properties of the current version.

The format of the vector of string vectors is:

```
[["property1", "value1"], ["property2", "value2"], ["propertyN", "valueN"]]
```

The property string is the name of the property. The value string specifies the value of the property.

Examples

This example returns the properties on version 3 of the design object "project".

```
$$get_version_properties("$PROJECT_XYZ/project", "Document_file", 3);
```

The preceding example returns the following vectors:

```
[["test", "test value"], ["engineer", "mary"], ["due_date", "2/3/92"]]
```

Related Topics

[\\$\\$delete_version\(\)](#)

[\\$\\$freeze_version\(\)](#)

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

[\\$\\$get_version_depth\(\)](#)

[\\$\\$revert_version\(\)](#)

[\\$\\$set_version_depth\(\)](#)

[\\$\\$set_version_property\(\)](#)

[\\$\\$unfreeze_version\(\)](#)

\$\$get_working_directory()

Scope: dme_dn_tk

Returns the value of the MGC working directory.

Usage

\$\$get_working_directory()

Arguments

None.

Return Values

The working directory or “NULL”.

Description

If the working directory has not been initialized, this function attempts to use the operating system's default value. This value is not always reliable. If the default value could cause you trouble, this function returns an error with a message directing you to set the working directory.

All Pyxis Project Manager windows, except for the Navigator window, use the value returned by the \$\$get_working_directory() function to resolve relative pathnames. The Navigator window uses the \$get_navigator_directory() function to resolve relative pathnames.

Examples

This example returns the MGC working directory.

```
$$get_working_directory();
```

Related Topics

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

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

[\\$get_navigator_directory_hard\(\)](#)

\$goto_directory()

Scope: dmgr_model

Prerequisite: To use this function, you must be in an active Navigator window.

Navigates to the specified directory.

Usage

`$goto_directory("directory_name")`

GOTo DDirectory directory_name

CD directory_name

Navigator window > **Explore** > **Go To**

Arguments

- **directory_name**

A string that specifies the directory to which the navigator goes. You may specify either an absolute or a relative pathname.

Return Values

VOID.

Description

Navigates to the directory specified by the `directory_name` argument. This function accepts both hard and soft pathnames as input. If you specify a directory that does not exist or a design object that is not a container, this function displays an error message and does not perform the navigation.

If you do not specify a `directory_name` argument, the `$goto_directory()` function displays a dialog box that prompts you for the new directory. The dialog box contains a stack of soft path locations that you have previously visited during this session. If a hard path cannot be resolved, the hard path is displayed in the dialog box.

To specify the directory to which you wish to navigate, you can enter a soft or hard directory pathname in the "Change directory to:" text input box and then run the dialog box. The navigator displays the soft path directory you specified, if possible. Alternatively, if you have previously navigated to the desired location, you can click on that pathname, as it is displayed in the dialog box, and then click the **OK** button to navigate to the desired directory.

You can use one of the command alias' or the GotoDirectory key as a short-cut method for navigating to a new directory, as shown in "[Key Names Mapped to Functions & Scopes](#)" on page 804.

Every time you navigate to a new directory, the number of columns needed to display icons in that directory is recalculated.

Examples

1. These examples use relative pathnames to navigate to a directory.

```
$goto_directory("../");
```

```
$goto_directory("examples");
```

2. This example uses an absolute pathname to navigate to a directory.

```
$goto_directory("$PROJ_DESIGN/brad/examples");
```

3. These examples show the command syntax from the popup command line.

```
got di ../
```

```
got di examples
```

```
got di $PROJ_DESIGN/brad/examples
```

Related Topics

[\\$explore_contents\(\)](#)

[\\$explore_references\(\)](#)

[\\$explore_parent\(\)](#)

\$\$handle_map_error()

Scope: dme_dn_tk

Enables the user to determine how to proceed when an error occurs in reading a location map.

Usage

```
$$handle_map_error("map_name")
```

Arguments

- **map_name**

A string that specifies the pathname of the location map.

Return Values

VOID.

Description

This function is called from \$read_map() when an error is encountered while reading a location map file. The \$\$handle_map_error() function causes a dialog box to appear, giving the user the following options:

- **Edit the location map file** — The location map file is opened in a Notepad window for editing.
- **Exit the application** — The \$close_session() is called to close the application session.
- **Continue** — The application session continues. If a valid location map was in memory prior to encountering the error, then that location map is still active; otherwise, the application session continues with no location map active.

Examples

This example shows how to use the \$\$handle_map_error() function in a routine that uses the \$read_map() function to read a location map. If \$read_map() returns a bad status code, this example checks first to see if the pathname to the location map file is valid and then checks to see if the location map file is protected. If the file exists and is not protected, then \$\$handle_map_error() is called.

```
function check_map(  
    file_name : string  
) , SEALED  
{  
    local status;  
    $$read_map(file_name);  
    status = $$get_status_code();  
    if (status != 0) {  
        $$report_global_status();  
        local type = $file_exist(file_name);  
        if (type == @file) {  
            if ($$is_read_protected(file_name, "Mgc_file") ==  
                @false && $$is_write_protected(file_name,  
                    "Mgc_file") == @false) {  
                $$handle_map_error(file_name);  
            }  
        }  
    }  
}
```

Related Topics

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

\$\$has_object_property()

Scope: dme_do_tk

Checks whether or not the specified property name and value exist on the specified design object.

Usage

```
$$has_object_property("obj_name", "obj_type", version, "prop_name", "prop_value")
```

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **obj_type**
A string that specifies the type of the design object.
- **version**
An integer that specifies the version number of the design object. The default is the current version.
- **prop_name**
A string that specifies the name of the property.
- **prop_value**
A string that specifies the value of the property.

Return Values

A Boolean that indicates whether the specified property exists on the specified design object. If the property exists, this function returns @true; if it does not exist, it returns @false. If an error occurs, this function returns UNDEFINED.

Description

When you set an object property, it remains on the current version of the design object. The object property carries forward as the design object evolves and the version number is incremented. However, if you set a property on a previous version, the property remains with that version and does not carry forward as the design object evolves.

By default, this function checks for the object property on the current version of the specified design object. To determine if the property exists on a previous version of the design object, you must specify the version argument.

Examples

This example checks if the “engineer” property and the value “John Smith” exist for version 2 of the “new_file” design object.

```
$$has_object_property("$PROJECT_XYZ/project/new_file",  
    "Image_file", 2, "engineer", "John Smith");
```

Related Topics

[\\$\\$delete_object_property\(\)](#)

[\\$\\$set_object_property\(\)](#)

[\\$\\$get_object_properties\(\)](#)

\$\$has_reference_property()

Scope: `dme_do_tk`

Checks whether or not a property exists on the specified reference.

Usage

```
$$has_reference_property("obj_name", "obj_type", obj_version, "target_obj_name",  
    "target_type", target_version, "prop_name", "prop_value")
```

Arguments

- **obj_name**
A string that specifies the pathname of the source design object.
- **obj_type**
A string that specifies the type of the source design object.
- ***obj_version***
An integer that specifies the version number of the source design object. The default is the current version.
- **target_obj_name**
A string that specifies the absolute of the target design object.
- **target_type**
A string that specifies the type of the target design object.
- ***target_version***
An integer that specifies the version number of the target design object. The `target_version` also represents the state of the reference. If the `target_version` is greater than 0, the state of the reference this function operates on is "fixed". If omitted or specified as 0, the state of the reference this function operates on is "current". The default is the current version.
- **prop_name**
A string that specifies the name of the property.
- ***prop_value***
A string that specifies the value of the property.

Return Values

A Boolean that indicates whether the specified property exists on a reference. If the reference property exists, this function returns `@true`; if it does not exist, it returns `@false`. If an error occurs, this function returns `UNDEFINED`.

Description

To check for a reference property on a previous version of the source design object, you must specify the “obj_version” argument.

Because other tools can create duplicate references, you can use the `$$has_reference_property_handle()` function for more accuracy when checking for reference properties.

If you specify the `target_version` argument, this function checks the properties of the first fixed reference found whose name, type, property, and version number match the specified arguments. As the `target_version` argument is specified, the state of the reference is fixed and the version number is checked.

If you omit the `target_version` argument, then this function checks the properties of the first current reference found whose name, type, and property match the specified arguments. As the `target_version` argument is not specified, the state of the reference is current and the version number is not checked.

If duplicate references exist, `$$has_reference_properties()` returns the properties of the first matching reference and a warning status, which states that more than one reference was found and that the properties returned may not be the properties that you intend to work with.

If you omit the `prop_value` argument, this function searches for a property whose name matches the `prop_name` argument, but it ignores property values.

Examples

This example checks if the property “test_status” and the value “complete” exists on the reference from version 3 of the “sample” source design object to version 2 of the “project” target design object. Because the `target_version` argument *is* specified, the reference is fixed and the version number is checked.

```
$$has_reference_property("$PROJECT_XYZ/plan/sample",  
    "Image_file", 3, "$PROJECT_XYZ/project", "Document_file",  
    2, "test_status", "complete");
```

Related Topics

[\\$\\$delete_reference_property\(\)](#)

[\\$\\$get_reference_properties\(\)](#)

[\\$\\$has_reference_property_handle\(\)](#)

[\\$\\$set_reference_property\(\)](#)

\$\$has_reference_property_handle()

Scope: dme_do_tk

Checks whether or not the specified property exists on the specified reference by using the target reference handle.

Usage

```
$$has_reference_property_handle("obj_name", "obj_type", obj_version, target_handle,  
    "prop_name", "prop_value")
```

Arguments

- **obj_name**
A string that specifies the pathname of the source design object.
- **obj_type**
A string that specifies the type of the source design object.
- **obj_version**
An integer that specifies the version number of the source design object. The default is the current version.
- **target_handle**
An integer that specifies the handle of the target design object.
- **prop_name**
A string that specifies the name of the property.
- **prop_value**
A string that specifies the value of the property.

Return Values

A Boolean that indicates if the specified property exists on the specified reference. If the reference property exists, this function returns @true; otherwise, this function returns @false.

Description

Checking for a reference property through the target handle enables you to distinctly specify which reference property to check. To find the reference's target handle, you use the `$$get_object_references()` function.

To check for a reference property on a previous version of the source design object, you must specify the `obj_version` argument.

If you omit the `prop_value` argument, this function searches for a property whose name matches the `prop_name` argument, and it ignores property values.

Examples

This example checks if the property “test_status” and the value “complete” exist on version 3 of the *sample* design object, using the reference handle 4.

```
$$has_reference_property_handle("$PROJECT_XYZ/sample",  
    "Image_file", 3, 4, "test_status", "complete");
```

Related Topics

[\\$\\$delete_reference_property\(\)](#)

[\\$\\$get_reference_properties\(\)](#)

[\\$\\$has_reference_property\(\)](#)

[\\$\\$set_reference_property\(\)](#)

[\\$\\$get_object_references\(\)](#)

\$hide_monitor()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active configuration monitor window.

Removes the active configuration's Monitor tab from view and redisplay the active Configuration tab in its place within the Configuration window.

Usage

\$hide_monitor()

HIDe MOnitor

Arguments

None.

Return Values

VOID.

Description

You can customize your session setup values to specify that the configuration monitor window is either visible or hidden during configuration operations.

Examples

1. This example hides the active configuration's monitor window.

\$hide_monitor();

2. This example hides the active configuration's monitor window by using command syntax from the popup command line.

hid mo

Related Topics

[\\$\\$clear_monitor\(\)](#)

[\\$show_monitor\(\)](#)

[\\$\\$writeln_monitor\(\)](#)

[Change Your Session Setup](#)

\$hide_secondary_entries()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window that currently shows secondary entries.

Hides the secondary entries in the active Configuration window.

Usage

`$hide_secondary_entries()`

HIDE SEcondary Entries

View > Hide Secondary Entries

Configuration window popup menu > **Hide Secondary Entries**

Arguments

None.

Return Values

VOID.

Description

You use this function to view only the primary entries. This function is only available from the menu if you are currently viewing primary entries.

Examples

1. This example hides the secondary entries in the active Configuration window.

```
$hide_secondary_entries();
```

2. This example hides the secondary entries in the active Configuration window by using command syntax from the popup command line.

```
hid se e
```

Related Topics

[\\$view_containment_hierarchy\(\)](#)

[\\$view_secondary_entries\(\)](#)

[\\$view_primary_hierarchy\(\)](#)

Chapter 4

Function Dictionary Part 3

Temporary short description for converting multi .fm chapter.

\$import_classic_data()

Scope: dmgr_proj_tk

Imports an individual component or a directory into a library, external library, or category.

Usage

```
$import_classic_data("source_data", "source_location_map", "destination_container",  
    lock_src_before_copy, follow_references, reference_vec, continue_on_error,  
    conflict_resolution)
```

File > Import > Classic Data

Arguments

- **source_data**
The path to the data to be imported.
- **source_location_map**
The location map used to resolve soft paths in the source data.
- **destination_container**
The path to the hierarchy design object that contains the imported data. This may be a hard path or soft path based on either an environment variable or an entry in the active location map at the time the function is called.
- **lock_src_before_copy**
A Boolean value indicating that the source data should be locked while the data is copied to the destination.
- **follow_references**
A Boolean value indicating that any data that is referenced by the design objects in source_data should also be imported.

References are only followed if the referenced data shares the same soft prefix as source_data, or if the referenced data begins with one of the soft prefixes listed in reference_vec.
- **reference_vec**
A vector of string values that match soft prefixes in the source location map. If follow_references is true, then any referenced data with a soft prefix listed in reference_vec are also copied to the destination.
- **continue_on_error**
A Boolean value indicating that the Pyxis Project Manager application should attempt to import as much data as possible even if errors occur. If true, and there is an error during the import operation, the Pyxis Project Manager application logs the error and attempts to continue, if possible. If false, the import operation aborts at the first error.

- **conflict_resolution**

A switch indicating desired behavior if conflict are found between the source and destination data. May be one of the following values: @overwrite, @abort, @prompt, or @skip.

Return Values

VOID.

Examples

The following example imports a directory from */tmp/dir* to the library at *\$PROJ/lib*. All referenced objects beginning with the *\$SRC_GEN* soft prefix are also copied. Source design objects are skipped if they conflict with existing design objects in the destination container.

```
$import_classic_data("/tmp/dir", "/tmp/dir/mgc_location_map", "$PROJ/lib", @false, @true, ["$SRC_GEN"], @false, @skip);
```

Related Topics

[\\$import_custom_view\(\)](#)

[\\$import_icstudio_library\(\)](#)

[\\$import_design_kit\(\)](#)

[\\$import_icstudio_project\(\)](#)

\$import_custom_view()

Scope: dmgr_proj_tk

Copies a custom view to a cell in a project, external library, or technology library.

Usage

```
$import_custom_view("view_path", "view_type", "dest_cell_path")
```

impcuv

IMPort CUsom View

File > Import > Custom View

Arguments

- **view_path**
The path to the custom view to be imported. The view path should not include the extension if that extension is associated with the view type.
- **view_type**
The type of the custom view. If no type is defined for this view, then this should be "Mgc_file".
- **dest_cell_path**
The path to the cell where the view is copied. This may be a hard path or soft path based on either an environment variable or an entry in the active location map at the time the function is called.

Return Values

VOID.

Examples

The following example imports a custom view from */tmp/view* to a cell within a project at */tmp/project/lib/cell*.

```
$import_custom_view("/tmp/view", "Mgc_file", "/tmp/project/lib/cell");
```

Related Topics

[\\$\\$add_type\(\)](#)

[\\$\\$update_type\(\)](#)

[\\$import_classic_data\(\)](#)

\$import_design_kit()

Scope: dmgr_proj_tk

Imports a design kit to produce a new technology library.

Usage

```
$import_design_kit("source_kit", "source_location_map", "destination_container",  
    "destination_name", lock_src_before_copy, continue_on_error)
```

impdek

IMPort DSign Kit

File > Import > Design Kit

Arguments

- **source_kit**
The path to the data to be imported.
- **source_location_map**
The location map used to resolve soft paths in the source data. Many design kit location maps rely on environment variables to define their hard paths. To import a design kit, either all environment variables must be set before invoking the Pyxis Project Manager application, or else all hard paths must be defined in the source location map.
- **destination_container**
The path to the directory that contains the new technology library.
- **lock_src_before_copy**
A Boolean value indicating that the source data should be locked while the data is copied to the destination.
- **continue_on_error**
A Boolean value indicating that the Pyxis Project Manager application should attempt to import as much data as possible even if errors occur. If true, and there is an error during the import operation, the Pyxis Project Manager application logs the error and attempt to continue, if possible. If false, the import operation aborts at the first error.

Return Values

VOID.

Examples

The following example takes a design kit from */tmp/kit* and creates a new technology library at */tmp/dest/new_tech_lib*.

```
$import_design_kit("/tmp/kit", "/tmp/kit/mgc_location_map", "/tmp/dest", "new_tech_lib",  
    @false, @false);
```

Related Topics

[\\$import_classic_data\(\)](#)

[\\$import_icstudio_project\(\)](#)

[\\$import_icstudio_library\(\)](#)

\$import_ext_lib()

Scope: dmgr_project_model

Creates a new project library which is a working copy of the specified external library within the destination project.

Usage

```
$import_ext_lib("ext_lib_path", "dest_project_path", "new_lib_name");
```

File > Import > External Library

Arguments

- **ext_lib_path**
The path to the external library which is to be imported.
- **dest_project_path**
The path to the project in which the new project library is created.
- **new_lib_name**
The name of the new library which is to be created.

Return Values

If the import operation is successful the return value is a string containing the full path to the newly created project library otherwise the return value is VOID.

Description

This function enables an external library to be specialized for use within a single project. The source external library is not modified by this operation.

Examples

The following example creates a copy of the */tmp/ext_lib* external library as a project library in the */tmp/project* project named "specialized".

```
$import_ext_lib("/tmp/ext_lib", "/tmp/project", "specialized");
```

Related Topics

[\\$create_dm_library\(\)](#)

[\\$export_library\(\)](#)

\$import_hdl()

Scope: lang_ifc

Imports an HDL file, compiles it, and generates symbols for each specified design unit.

Usage

```
$import_hdl("hdl_do_path", "hdl_do_type", [du_names], "destination_path", [open_ports],  
            [open_pins], use_default_options, compiler, "adms_options", "other_options", includes,  
            "hdl_override_type", [symbol_paths], add_missing_symbol_props, props_visibility)
```

File > Import > HDL

Arguments

- **hdl_do_path**
A string value that specifies the path to the HDL source file, without the file extension.
- **hdl_do_type**
A string value that specifies the type of the HDL source file. Valid types are: "mgc_vhdl", "mgc_vhdl_ams", "mgc_verilog", "mgc_verilog_a", and "mgc_verilog_ams".
- **du_names**
A vector of strings that specify the names of the Verilog modules or VHDL entities to generate symbols for.
- **destination_path**
A string that specifies the path to the Component where the HDL source file is copied. If the component does not exist, it is created.
- **open_ports**
A vector of strings that specifies the names of ports that may be on the model but do not display on the generated symbol.
Default: [].
- **open_pins**
A vector of vectors that specifies the ports that may not be on the model but may display on the generated symbol.
Expected format: [{"name", implicit, "direction", "signal_type", "pin_type"}, ...].
Default: [].
A description of the elements is as follows:
 - **name** — A string that specifies the name of the symbol pin.
 - **implicit** — A Boolean that specifies whether the pins are implicit (@true) or not implicit (@false).

- **direction** — A string that specifies the direction of the symbol pin. Valid values are “input”, “output”, or “bidir”.
- **signal_type** — A string that specifies the type of the signal in the HDL model.
- **pin_type** — A string that specifies the type of the pin in the HDL model.
- **use_default_options**

A Boolean value indicating that the HDL should be compiled according to the default compilation options for the current hierarchy. Options include:

 - **@true** — The remaining arguments are ignored. The file is compiled based on the default compilation options for the project. Default.
 - **@false** — The file is compiled according to the other specified options, and those options are saved as the imported file's compilation options, (which is similar to calling the \$set_file_compilation_options function).
- **compiler**

Specifies the compiler to use. Options include: @eldo, @adms, or @questa.

Not all options are valid for all HDL types. For Verilog and VHDL, valid options are @adms and @questa. For Verilog A, valid options are @adms and @eldo. For Verilog AMS and VHDL-AMS, only @adms is available.
- **adms_options**

A string value containing any extra flags to be passed to the ADMS compiler. Default: “”.
- **other_options**

For Verilog and VHDL, specifies a string value containing any extra flags to pass to the Questa compiler. For Verilog A, specifies a string value containing any extra flags to pass to the Eldo compiler. Default: “”.
- **includes**

For Verilog, Verilog A, and Verilog AMS, specifies a vector of strings that contain include paths to pass to the compilers. Default: [].
- **hdl_override_type**

A string that determines whether or not you import the HDL file as the same type as the original file. The hdl_override_type can have the following possible values: “mgc_system_verilog”, “mgc_vhdl”, “mgc_vhdl_ams”, “mgc_verilog”, “mgc_verilog_a”, and “mgc_verilog_ams”.

Default: “”, which specifies that the hdl_override_type is the same value as the hdl_do_type setting.

You can import the original file in the type specified by the hdl_override_type value if its value is compatible with hdl_do_type. The combinations of compatible types for the hdl_override_type and hdl_do_type values are:

- System Verilog, Verilog, Verilog-A, and Verilog-AMS.
- VHDL and VHDL-AMS.
- ***symbol_paths***

A vector of strings. Each contains the path to a symbol that should be associated with the design unit specified by the `du_names` argument. The order and number of this argument must also match the `du_names` argument.

Default: [].
- ***add_missing_symbol_props***

A Boolean indicating whether or not the application should add the missing symbol properties to existing symbols.

Default: @false.
- ***props_visibility***

A name that controls the symbol property visibility on generated symbols. It can take on one of the following three values:

 - **@show_all** — Makes all properties visible. Default.
 - **@hide_all** — Hides all properties.
 - **@customize** — Iteratively invokes the [Symbol Layout Properties Dialog Box](#) for each model enabling you to customize the visibility of each property on the model.

Return Values

VOID.

Examples

1. The following example imports the *delay_block.v* HDL file to the given *\$TEST_FLOW/flow_lib/new_delay_block* destination path, compiles according to the default compilation options for the current hierarchy with the ADMS compiler, and generates symbols for given "delay_block" design unit. (No values for "open_ports" and "open_pins" are specified). The property text and value are visible on the symbol.

```
$import_hdl("$TESTDIR/sources/delay_block", "mgc_verilog", ["delay_block"],  
"$TEST_FLOW/flow_lib/new_delay_block", [], [], @true, @adms, "", "", [],  
"mgc_verilog", ["$TEST_FLOW/flow_lib/new_delay_block/delay_block"], @true,  
@show_all);
```

2. The following example creates the new *inv.va* file in *\$PROJ_GENERIC13/test_libA/inv*. (Note that the Verilog file is imported as a Verilog-A file).

```
$import_hdl("$PROJ_GENERIC13/test_libA/inv/inv", "mgc_verilog", ["inv"],  
"$PROJ_GENERIC13/test_libA/inv", [], [], @true, @adms, "", "", [], "mgc_verilog_a",  
["$PROJ_GENERIC13/test_libA/inv/inv"], @true, @show_all);
```

Related Topics

[\\$import_spice\(\)](#)

[\\$set_file_compilation_options\(\)](#)

\$import_icstudio_library()

Scope: dmgr_proj_tk

Imports an ICstudio library to produce a new Pyxis Project Manager external library.

Usage

```
$import_icstudio_library("source_library", "destination_container", "destination_name",  
    "tech_config_path", lock_src_before_copy, continue_on_error)
```

impicl

IMPort ICstudio Library

File > Import > ICstudio Library

Arguments

- **source_library**
The path to the ICstudio library to be imported. The library should be contained in an ICStudio project and included in that project's library list.
- **destination_container**
The path to the directory that contains the new external library.
- **destination_name**
An optional string value specifying the name of the new external library. If no name is given, the name of the source library is used.
- **tech_config_path**
The path to the technology configuration to use in the new external library.
- **lock_src_before_copy**
A Boolean value indicating that the source data should be locked while the data is copied to the destination.
- **continue_on_error**
A Boolean value indicating that the Pyxis Project Manager application should attempt to import as much data as possible even if errors occur. If true, and there is an error during the import operation, the Pyxis Project Manager application logs the error and attempt to continue, if possible. If false, the import operation aborts at the first error.

Return Values

VOID.

Examples

The following example imports the */tmp/A.proj/B.lib* ICstudio library to a new Pyxis Project Manager external library at */tmp/dest/B*. The new external library uses the technology library at */tmp/tech_lib* along with the "my_tech_config" technology configuration.

```
$import_icstudio_library("/tmp/A.proj/B.lib", "/tmp/dest", "",  
"/tmp/tech_lib/configurations/my_tech_config", @false, @true);
```

Related Topics

[\\$import_classic_data\(\)](#)

[\\$import_icstudio_project\(\)](#)

[\\$import_design_kit\(\)](#)

\$import_icstudio_project()

Scope: dmgr_proj_tk

Imports an ICstudio project to produce a new Pyxis Project Manager project.

Usage

```
$import_icstudio_project("source_project", "destination", "tech_config_name",  
    "technology_library", lock_src_before_copy, continue_on_error, [included_libraries])
```

impicp

IMPort ICstudio Project

File > Import > ICstudio Project

Arguments

- **source_project**
The path to the ICstudio project to be imported.
- **destination**
The path to the new project.
- **tech_config_name**
The name of the technology configuration to use in the new project. The technology configuration must exist in the technology library indicated by "technology_library".
- **technology_library**
The path to the technology library to use in the new project.
- **lock_src_before_copy**
A Boolean value indicating that the source data should be locked while the data is copied to the destination.
- **continue_on_error**
A Boolean value indicating that the Pyxis Project Manager application should attempt to import as much data as possible even if errors occur. If true, and there is an error during the import operation, the Pyxis Project Manager application logs the error and attempts to continue, if possible. If false, the import operation aborts at the first error.
- **included_libraries**
A vector of string values, which are paths to external ICstudio libraries. These libraries are imported to the same destination directory as the project. Any references between the design objects in the project and the external libraries are updated during the import operation.

Return Values

VOID.

Examples

The following example imports the */tmp/A.proj* ICstudio project to a new Pyxis Project Manager project at */tmp/dest/A*. The new project uses the technology library at */tmp/tech_lib* along with the “my_tech_config” technology configuration.

```
$import_icstudio_project("/tmp/A.proj", "/tmp/dest", "my_tech_config", "/tmp/tech_lib",  
@false, @true, []);
```

Related Topics

[\\$import_classic_data\(\)](#)

[\\$import_icstudio_library\(\)](#)

[\\$import_design_kit\(\)](#)

\$import_spice()

Scope: lang_ifc

Imports a SPICE file and generates symbols for each subcircuit.

Usage

```
$import_spice("spice_do_path", "spice_do_type", [subcircuit_names], "destination_path",  
             inline_includes, [open_ports], [open_pins], spice_types, [symbol_path],  
             add_missing_symbol_props, "props_visibility")
```

File > Import > SPICE

Arguments

- **spice_do_path**
A string value that specifies the path to the SPICE source file, without the file extension.
- **spice_do_type**
A string value that specifies the exact DDMS type of the source file, such as "mgc_spice_spi" or "mgc_spice_cir".
- **subcircuit_names**
A vector of strings, where each string is the name of a subcircuit in the source file.
- **destination_path**
A string that specifies the path to the component that will contain the imported source file. If the component does not exist, it is created.
- **inline_includes**
A Boolean which if @true, then for any SPICE files that were included by the source file, the contents of the included files are inserted in place of the ".INCLUDE" lines in the imported file.
Default: @true.
- **open_ports**
A vector of strings that specifies the names of ports that may be on the model but are not included in the generated symbols. Default: [].
- **open_pins**
A vector of vector strings that specifies the ports that may not be on the model but are included in the generated symbols.
Expected format: [{"name", bool_implicit, "direction", "signal_type", "pin_type"}, ...].
Default: [].
A description of the elements is as follows:
 - **name** — A string that specifies the name of the symbol pin.

- **bool_implicit** — A Boolean that specifies whether the pins are implicit (@true) or not implicit (@false).
- **direction** — A string that specifies the direction of the symbol pin. Valid values are “input”, “output”, or “bidir”.
- **signal_type** — A string that specifies the type of the signal in the SPICE model.
- **pin_type** — A string that specifies the type of the pin in the SPICE model.
- ***spice_types***
A vector of four Boolean values that specifies the SPICE language to use in NCF.
Expected format: [SPICE, EldoSPICE, HSPICE, LVS].
Default: [@true, @false, @false, @false].
- ***symbol_path***
A vector of strings. Each contains the path to a symbol that should be associated with the subcircuit specified by the subcircuit_names argument. The order and number of this argument must also match the subcircuit_names argument.
Default: [].
- ***add_missing_symbol_props***
A Boolean indicating whether or not the application should add the missing symbol properties to existing symbols.
Default: @false.
- ***props_visibility***
A name that controls the symbol property visibility on generated symbols. It can take on one of the following three values:
 - **@show_all** — Makes all properties visible. Default.
 - **@hide_all** — Hides all properties.
 - **@customize** — Prompts you with a list of properties and enables you to select the visibility of each.

Return Values

VOID.

Examples

The following example imports the SPICE file *\$PYXIS_SPT/GenericPLL/Simulations/FreqSynth/tb_FreqSynth/QuestaADMS/netlist* of type *mgc_spice_spi*, (indicating that the file ends with a *.spi* extension). The file contains two sub-circuits—FILTER and FREQSYNTH. The actual file is copied into the component *\$PYXIS_SPT/MixedSims/tb*. The two sub-circuits are registered with the symbols

\$PYXIS_SPT/MixedSims/FILTER/FILTER and
\$PYXIS_SPT/MixedSims/FREQSYNTH/FREQSYNTH respectively.

```
$import_spice("$PYXIS_SPT/GenericPLL/Simulations/FreqSynth/tb_FreqSynth/QuestADMS  
/netlist", "mgc_spice_spi", ["FILTER", "FREQSYNTH"], "$PYXIS_SPT/MixedSims/tb", @true,  
[], [], [@true, @false, @false, @false], ["$PYXIS_SPT/MixedSims/FILTER/FILTER",  
"$PYXIS_SPT/MixedSims/FREQSYNTH/FREQSYNTH"], @true, @show_all);
```

Related Topics

[\\$import_hdl\(\)](#)

\$include_external_library()

Scope: dmgr_proj_tk

Adds a reference from a root hierarchical design object (HDO) such as a project or external library, to an external library, and includes the external library in the root HDO's location map.

Usage

```
$include_external_library("root_hdo_path", "ext_lib_path")
```

incexl

INClude EXternal Library

Edit > External/Logic Libraries

Arguments

- **root_hdo_path**
The path to the project or external library that should include the external library at "ext_lib_path".
- **ext_lib_path**
The path to the external library to be included.

Return Values

VOID.

Examples

The following example adds the */tmp/extlib* external library as an include to the */tmp/project* project:

```
$include_external_library("/tmp/project", "/tmp/extlib");
```

Related Topics

[\\$remove_external_library\(\)](#)

\$invoke_bgd_tool()

Scope: dm_tool_tk

Prerequisite: This function is only available from within a qualification or termination script.

Invokes an executable file as a background process.

Usage

```
$invoke_bgd_tool("executable", "other_argsN")
```

Arguments

- **executable**
A string that specifies the pathname of an executable file.
- **other_argsN**
A rest string that provides arguments to the executable file. Zero or more arguments are allowed.

Return Values

VOID.

Description

The tool viewpoint function `$invoke_bgd_tool()` invokes an executable file as a background process, which is a process that runs without a shell window open. This function is useful for starting background processes such as servers.

In addition to the executable argument, you can use any number of optional arguments. This function treats these additional arguments as arguments to the executable file and passes them through unchanged.

You can use this function in your tool viewpoint qualification script to invoke the actual tool that the tool viewpoint represents. This function also sets the active tool viewpoint to "NULL". You are only allowed to call this function once from within a qualification script.

The `$invoke_bgd_tool()` function passes the value of the `MGC_WD` and `MGC_LOCATION_MAP` environment variables to the invoked tool. In this way, the newly invoked tool inherits the Pyxis Project Manager current working directory and location map.

Examples

This example invokes a tool as a background process.

```
$invoke_bgd_tool("$PROJECT_XYZ/design/start_server", startup,  
                chktime, flag);
```

Related Topics

[\\$get_tool_script\(\)](#)

[\\$invoke_tool\(\)](#)

\$invoke_tool()

Scope: dm_tool_tk

Prerequisite: This function is only available from within a qualification or termination script.

Opens a new shell window and invokes the specified executable file.

Usage

`$invoke_tool("executable", "other_argsN")`

Arguments

- **executable**
A string that specifies the pathname of an executable file.
- **other_argsN**
A rest string that provides an argument to the executable file. Zero or more arguments are allowed. A comma must delimit each optional argument from the next.

Return Values

VOID.

Description

The tool viewpoint function `$invoke_tool()` opens a new shell window and executes the file specified by the executable argument. This file can be any executable file.

You can use this function in your tool viewpoint qualification script to invoke the actual tool that the tool viewpoint represents. This function also sets the active tool viewpoint to "NULL". You are only allowed to call this function once from within a qualification script.

In addition to the executable argument, you can use any number of optional arguments. This function treats these optional arguments as arguments to the executable file and passes them to the actual tool unchanged. Each of the optional arguments must be separated from the next optional argument by a comma.

This function cannot specify the dimensions or the location of the new shell window. In addition, the shell window does not close after the tool session terminates. Because the shell window is left open, you can recover the transcript from the design tool session.

The `$invoke_tool()` function passes the value of the `MGC_WD` and `MGC_LOCATION_MAP` environment variables to the invoked tool. In this way, the newly invoked tool inherits the Pyxis Project Manager current working directory and location map.

When you invoke a tool with this function, a process manager records all of the information needed to tell the Pyxis Project Manager application to run the termination script (if it exists) when the tool session quits.

Examples

This example invokes the *\$MGC_HOME/bin/dss* tool with optional arguments.

```
$invoke_tool("$MGC_HOME/bin/dss", sheet, startflag, startupfile,  
installflag, installfile, titleflag, titlestr, pkgflag, pkgstr);
```

Related Topics

[\\$get_tool_script\(\)](#)

[\\$invoke_bgd_tool\(\)](#)

\$\$is_build_consistent()

Scope: dm_config_tk

Checks whether or not the active configuration is consistent.

Usage

\$\$is_build_consistent()

Arguments

None.

Return Values

A Boolean that indicates if the build is consistent. If the build is consistent, this function returns @true; otherwise, it returns @false. If an error occurs, this function returns UNDEFINED.

Description

A configuration is consistent if it is locked and a build has been performed. A consistent configuration ensures the integrity of the design object versions that are part of the configuration. You may want to check if a configuration is consistent prior to releasing or copying it.

To create a configuration that is consistent, you first lock the configuration using the \$\$lock_configuration() function, and then build the configuration using the \$\$build_configuration() function. Once you close the configuration, the lock is removed and the configuration is no longer consistent.

Examples

This example opens the “config_x” configuration and checks if it is consistent. If the configuration is consistent, it is then copied.

```
$$open_configuration("/user/jane/project/config_x",0,@read_write;  
  
local consistent = $$is_build_consistent();  
  
if (consistent == true) {  
  
  $$copy_configuration("/user/mark/misc", @replace, @create); }
```

Related Topics

[\\$\\$is_build_valid\(\)](#)

[\\$\\$lock_configuration\(\)](#)

[\\$\\$build_configuration\(\)](#)

\$\$is_build_valid()

Scope: dm_config_tk

Checks whether or not the active configuration is valid.

Usage

\$\$is_build_valid()

Arguments

None.

Return Values

A Boolean that indicates whether a configuration build is valid. If the configuration build is valid, this function returns @true. Otherwise, it returns @false. If an error occurs, this function returns UNDEFINED.

Description

A configuration is valid after a build is performed. It remains valid until a change is made to the configuration. For example, if you add or remove an entry *after* you have performed a build, the configuration is no longer valid.

You may want to check if a configuration is valid prior to releasing or copying it, to ensure that the released or copied configuration has not changed since the last build.

Examples

This example opens the “config_a” configuration and checks if it is valid. If the configuration is valid, it is released.

```
$$open_configuration("$PROJECT_XYZ/config_a", 0,  
    @read_write);  
  
local valid = $$is_build_valid();  
  
if (valid == true) {  
    $$release_configuration("$PROJECT_XYZ/release_dir",  
        "release2", @create_dest); }
```

Related Topics

[\\$\\$is_build_consistent\(\)](#)

[\\$\\$is_configuration_edited\(\)](#)

\$\$is_configuration_edited()

Scope: dm_config_tk

Prerequisite: To use this function, you must have an active configuration.

Checks whether or not the active configuration has been changed and needs to be saved.

Usage

`$$is_configuration_edited()`

Arguments

None.

Return Values

A Boolean that indicates whether the active configuration has changed. If the configuration has been edited, this function returns `@true`. Otherwise, it returns `@false`. If an error occurs, this function returns `UNDEFINED`.

Description

The toolkit function `$$is_configuration_edited()` checks if a configuration has been changed since the last save operation. This function is useful in indicating when a configuration object needs to be saved within the context of a configuration edit session.

This function detects all changes made to the configuration object by a configuration function. Changes made to the configuration object by a configuration function include operations such as adding or removing an entry, changing a build rule or an entry's target path, and building the configuration; this function detects these operations as changes. However, changes made by functions that treat the configuration as an ordinary design object, such as changing, removing, reverting or adding references, do not qualify as changes to the configuration and are not detected by this function.

Examples

This example checks if the active configuration is changed.

```
$$is_configuration_edited();
```

Related Topics

[\\$\\$is_build_valid\(\)](#)

\$\$is_configuration_frozen()

Scope: dm_config_tk

Prerequisite: To use this function, you must have an active configuration.

Checks whether or not the active configuration is frozen.

Usage

\$\$is_configuration_frozen()

Arguments

None.

Return Values

A Boolean that indicates whether the active configuration is frozen. If the configuration is frozen, this function returns @true. Otherwise, it returns @false. If an error occurs, this function returns UNDEFINED.

Examples

This example checks if the active configuration is frozen.

```
$$is_configuration_frozen();
```

Related Topics

[\\$\\$freeze_configuration\(\)](#)

[\\$\\$unfreeze_configuration\(\)](#)

\$\$is_configuration_locked()

Scope: dm_config_tk

Prerequisite: To use this function, you must have an active configuration.

Checks whether or not the active configuration is locked.

Usage

\$\$is_configuration_locked()

Arguments

None.

Return Values

A Boolean that indicates whether the active configuration is locked. If the configuration is locked, this function returns @true. Otherwise, it returns @false. If an error occurs, this function returns UNDEFINED.

Examples

This example checks if the active configuration is frozen.

```
$$is_configuration_locked();
```

Related Topics

[\\$\\$lock_configuration\(\)](#)

[\\$\\$unlock_configuration\(\)](#)

\$\$is_container()

Scope: dme_do_tk

Checks whether or not the specified design object is a container.

Usage

```
$$is_container("obj_name", "type")
```

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **type**
A string that specifies the type of the design object. The default is "Mgc_container".

Return Values

A Boolean that indicates whether the specified design object is a container. If the specified design object is a container, this function returns @true. If it is not a container, it returns @false. If an error occurs, this function returns UNDEFINED.

Description

This function searches the fileset of the design object and checks if it contains at least one container. A container is a file system directory or has a file system directory as a member of its fileset.

Examples

This example checks whether or not the "base" design object is a container object.

```
$$is_container("$PROJECT_XYZ/base", "Mgc_container");
```

Related Topics

[\\$\\$get_container_contents\(\)](#)

[\\$\\$is_directory\(\)](#)

\$\$is_directory()

Scope: dme_do_tk

Checks whether or not the specified design object is a directory.

Usage

```
$$is_directory("obj_name")
```

Arguments

- **obj_name**

A string that specifies the pathname of the design object.

Return Values

A Boolean that indicates whether or not the specified design object is a directory. If the design object is a directory, this function returns @true. Otherwise, it returns @false.

Description

This function only checks if the specified design object is a file system directory; it does not check the directory's fileset members.

Examples

This example checks whether or not the "project" design object is a directory.

```
$$is_directory("$PROJECT_XYZ/project");
```

Related Topics

[\\$\\$add_directory\(\)](#)

[\\$\\$is_container\(\)](#)

\$\$is_entry_container()

Scope: dm_config_tk

Checks whether or not the specified configuration entry is a container.

Usage

`$$is_entry_container("name", "type", version)`

Arguments

- **name**
A string that specifies the pathname of the configuration entry.
- **type**
A string that specifies the type of the configuration entry.
- ***version***
An integer that specifies the version number of the configuration entry. The default is 0, which represents the current version.

Return Values

A Boolean that indicates whether the specified configuration entry is a container. If the specified entry is a container, this function returns @true. Otherwise, it returns @false. If an error occurs, this function returns UNDEFINED.

Description

An entry is a container if it can contain other design objects.

Examples

This example opens the “my_config” configuration and then checks if the “zoo” entry is a container.

```
$$open_configuration("$PROJECT_XYZ/d1/my_config",0,@read_write);  
  
local container = $$is_entry_container("$PROJECY_XYZ/project/zoo",  
    "mgc_sheet");
```

Related Topics

[\\$\\$get_children\(\)](#)

\$\$is_entry_fixed()

Scope: dm_config_tk

Checks whether or not the specified configuration entry is a fixed entry.

Usage

`$$is_entry_fixed("name", "type", version)`

Arguments

- **name**
A string that specifies the pathname of the configuration entry.
- **type**
A string that specifies the type of the configuration entry.
- ***version***
An integer that specifies the version number of the configuration entry. The default is 0, which represents the current version.

Return Values

A Boolean that indicates whether the specified configuration entry is fixed. If the specified entry is fixed, this function returns @true. Otherwise, it returns @false. If an error occurs, this function returns UNDEFINED.

Description

A fixed entry always points to the same version of a design object. When the Pyxis Project Manager application comes to a fixed entry during a build, all objects that are referenced by the fixed entry are added to the configuration as fixed entries.

Examples

This example opens the “config_beta” configuration and assigns the value of `$$is_entry_fixed()` to a variable named “fixed” for the “d1” entry.

```
$$open_configuration("$PROJECT_XYZ/project/config_beta", 0,  
    @read_write);  
  
local fixed = $$is_entry_fixed("$PROJECT_XYZ/d1", "Mgc_file", 0);
```

Related Topics

[\\$\\$get_reference_traversal\(\)](#)

[\\$\\$set_reference_traversal\(\)](#)

[\\$\\$is_entry_primary\(\)](#)

\$\$is_entry_primary()

Scope: dm_config_tk

Checks whether or not the specified configuration entry is a primary entry.

Usage

`$$is_entry_primary("name", "type", version)`

Arguments

- **name**
A string that specifies the pathname of the configuration entry.
- **type**
A string that specifies the type of the configuration entry.
- ***version***
An integer that specifies the version number of the configuration entry.
Default: 0, which represents the current version.

Return Values

A Boolean that indicates whether the specified configuration entry is a primary entry. If the configuration entry is a primary entry, this function returns @true. Otherwise, it returns @false. If an error occurs, this function returns UNDEFINED.

Description

A primary entry is an entry that you add to a configuration by using the `$$add_configuration_entry()` function.

Examples

This example opens the “config_alpha” configuration and then checks if the “plans” entry is a primary entry. If the entry is a primary entry, its filter is set so that it excludes objects of the “Dss_sheet” type from the configuration.

```
$$open_configuration("$PROJECT_XYZ/design/config_alpha", 0,  
    @read_write);  
  
local primary = $$is_entry_primary("$PROJECT_XYZ/d2/plans",  
    "Mgc_container", 0);  
  
if (primary == true) {  
    $$set_object_type_filter(["Dss_sheet"], ["exclude"]); }  
}
```

Related Topics

[\\$\\$get_reference_traversal\(\)](#)

[\\$\\$is_entry_fixed\(\)](#)

[\\$\\$add_configuration_entry\(\)](#)

\$\$is_entry_retargetable()

Scope: dm_config_tk

Checks whether or not the specified configuration entry is retargetable.

Usage

```
$$is_entry_retargetable("name", "type", version)
```

Arguments

- **name**
A string that specifies the pathname of the configuration entry.
- **type**
A string that specifies the type of the configuration entry.
- **version**
An integer that specifies the version number of the configuration entry. The default is 0, which represents the current version.

Return Values

A Boolean that indicates whether the specified configuration entry is retargetable. If the configuration entry is retargetable, this function returns @true. Otherwise it returns @false. If an error occurs, this function returns UNDEFINED.

Description

An entry is retargetable if its parent by containment is not part of the active configuration. If an entry is retargetable, you can specify its target path.

Examples

This example opens the “my_config” configuration and then checks if the “x” configuration entry is retargetable. If the entry is retargetable, the target path is set with the [\\$\\$set_target_path\(\)](#) function.

```
$$open_configuration("$PROJECT_XYZ/project/my_config",0,  
    @read_write);  
  
local retarget = $$is_entry_retargetable("$PROJECT_XYZ/x",  
    "Mgc_file, 0);  
  
if (retarget == true) {  
    $$set_target_path("$PROJECT_XYZ/project/release_dir2",  
        "$PROJECT_XYZ/x", "Mgc_file, 0); }
```

Related Topics

[\\$\\$get_target_path\(\)](#)

[\\$\\$set_target_path\(\)](#)

\$\$is_object_released()

Scope: dme_do_tk

Checks whether or not the specified design object is in a released state.

Usage

`$$is_object_released("obj_name", "type")`

Arguments

- **obj_name**
A string that specifies the pathname of the design object to be checked.
- **type**
A string that specifies the type of the design object to be checked.

Return Values

A Boolean that indicates whether the specified design object is in a released state. If the design object is in a released state, this function returns @true. Otherwise, it returns @false. If an error occurs, this function returns UNDEFINED.

Description

A design object is in a released state if it has been released using one of the release commands available from either the Navigator window or the Configuration window.

Examples

This example creates the \$edit_object() sample function that checks the release status of an object before invoking a tool or performing some type of edit function on the object.

```
function $edit_object(  
    obj_name : string,  
    obj_type : string  
)  
{  
    local released_state;  
    released_state = $$is_object_released(obj_name, obj_type);  
    if (released_state == @false) {  
        $open_editor(obj_name, obj_type);  
    }  
    else {  
        $message($strcat("Permission to edit denied: object ",obj_name," is  
        released"));  
    }  
}
```

Related Topics

[\\$\\$is_read_protected\(\)](#)

[\\$\\$object_complete\(\)](#)

\$\$is_writable()

\$\$object_exists()

\$\$is_write_protected()

\$\$is_object_versioned()

Reports whether or not the object is versioned.

Usage

`$$is_object_versioned(obj_name, obj_type)`

Arguments

- **obj_name**
A string that specifies the pathname to the object.
- **obj_type**
A string that specifies the object type.

Return Values

Returns either @true if the object is versioned; returns @false otherwise.

Examples

```
$$is_object_versioned("/usr2/maintain_3/pld_lib/fpld/part",  
    "Eddm_part");  
  
// Returns @true
```

Related Topics

[\\$\\$create_versioned_object\(\)](#)

[\\$\\$is_type_versioned\(\)](#)

\$\$is_read_protected()

Scope: dme_do_tk

Checks whether or not the specified design object can be opened for reading.

Usage

```
$$is_read_protected("obj_name", "type")
```

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **type**
A string that specifies the type of the design object.

Return Values

A Boolean that indicates whether the specified design object is read protected. If the design object is read protected, this function returns @true. If it is not, this function returns @false. If an error occurs, this function returns UNDEFINED.

Examples

This example checks if the “my_folder” design object is read protected.

```
$$is_read_protected("$PROJECT_XYZ/project/my_folder",  
    "Mgc_container");
```

Related Topics

[\\$\\$get_object_protection\(\)](#)

[\\$\\$set_protection\(\)](#)

[\\$\\$is_writable\(\)](#)

[\\$\\$set_protection_numeric\(\)](#)

[\\$\\$is_write_protected\(\)](#)

\$\$is_relative_path()

Scope: dme_dn_tk

Checks whether or not the specified pathname is a relative pathname.

Usage

```
$$is_relative_path("path_name")
```

Arguments

- **path_name**
A string that contains any pathname.

Return Values

A Boolean that indicates if the specified pathname is relative. If the pathname is relative, this function returns @true; otherwise, this function returns @false.

Description

If the specified pathname begins with a dollar sign (\$) or a slash (/), this function returns @false. If the specified pathname begins with any other character, this function returns @true.

Examples

1. This example returns @true, indicating that the *design2* pathname is relative.

```
$$is_relative_path("design2");
```
2. This example returns @false, indicating that the *\$MGC_GENLIB* pathname is not relative.

```
$$is_relative_path("$MGC_GENLIB");
```
3. This example returns @false, indicating that the */usr2/project/xyz/design1* pathname is not relative.

```
$$is_relative_path("/usr2/project/xyz/design1");
```

Related Topics

\$\$fix_relative_path()	\$\$get_soft_name()
\$\$get_hard_name()	\$\$get_working_directory()
\$get_navigator_directory()	\$\$set_working_directory()
\$get_navigator_directory_hard()	

\$\$is_type_versioned()

Checks whether or not the object type is versioned.

Usage

\$\$is_type_versioned(type_name)

Arguments

- **type_name**
The type name for the object.

Return Values

Returns either @true if the object type is versioned; returns @false otherwise.

Examples

```
$$is_type_versioned("Eddm_part");  
  
// Returns @true
```

Related Topics

[\\$\\$create_versioned_object\(\)](#)

[\\$\\$is_object_versioned\(\)](#)

\$\$is_writable()

Scope: dme_do_tk

Checks whether or not the current process can write to the specified design object.

Usage

```
$$is_writable("obj_name", "type")
```

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **type**
A string that specifies the type of the design object.

Return Values

A Boolean that indicates if the specified design object is writable. If the design object is writable, this function returns @true; otherwise, this function returns @false. If an error occurs, this function returns UNDEFINED.

Examples

This example checks if the “goals” design object is writable.

```
$$is_writable("$PROJECT_XYZ/project/goals", "Mgc_file");
```

Related Topics

[\\$\\$get_object_protection\(\)](#)

[\\$\\$set_protection\(\)](#)

[\\$\\$is_read_protected\(\)](#)

[\\$\\$set_protection_numeric\(\)](#)

[\\$\\$is_write_protected\(\)](#)

\$\$is_write_protected()

Scope: dme_do_tk

Checks whether or not the specified design object can be opened for writing.

Usage

```
$$is_write_protected("obj_name", "type")
```

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **type**
A string that specifies the type of the design object.

Return Values

A Boolean that indicates if the specified design object is write-protected. If the design object is write-protected, this function returns @true. If it is not, this function returns @false. If an error occurs, this function returns UNDEFINED.

Examples

This example checks if the “plans” design object can be opened for writing.

```
$$is_write_protected("$PROJECT_XYZ/project/plans","Mgc_file");
```

Related Topics

[\\$\\$get_object_protection\(\)](#)

[\\$\\$set_protection\(\)](#)

[\\$\\$is_read_protected\(\)](#)

[\\$\\$set_protection_numeric\(\)](#)

[\\$\\$is_writable\(\)](#)

\$list_references()

Provides an alphabetized list of all unique references in a design directory tree.

Usage

`$list_references()`

Arguments

None.

Return Values

VOID.

Description

Select one or more container objects in the Navigator, then type `$list_references()` in a popup command line. When the function completes, a report window displays containing the list of unique references for each selected object and grouped by the selected object. This information can serve a number of purposes, such as enabling you to quickly identify references you know may not resolve correctly, or by highlighting references that do not begin with soft prefixes.

Examples

`$list_references()`

Related Topics

[\\$find_references\(\)](#)

\$load_registry()

Scope: dmgr_type_area

Prerequisite: To use this function, you must be in an active type window.

Loads a type registry into the Type Manager window of the current session.

Usage

`$load_registry("pathname")`

LOAd REgistry pathname

File > Load Registry

Arguments

- **pathname**

A string that specifies the location of a type registry.

Return Values

VOID.

Description

The interactive function `$load_registry()` loads the types contained in the type registry into the Type Manager window of the current session. To load the registry, you must specify the "pathname" argument. If conflicting types exist in the type window, then this function replaces the conflicting types. As a result of loading your type registry, the Pyxis Project Manager application can recognize design objects and tools described in your type registry.

To create a type registry, use the Mentor Graphics Pyxis Registry application.

Examples

1. This example loads the registry specified by the pathname argument. The *\$MARY* soft prefix represents a sample user directory.

```
$load_registry("$MARY/my_registry.rgy");
```

2. This example loads the same registry by using command syntax from the popup command line.

```
loa re $MARY/my_registry.rgy
```

Related Topics

[\\$open_types_window\(\)](#)

[\\$report_type_info\(\)](#)

[Types](#)

\$\$lock_configuration()

Scope: dm_config_tk

Locks all design object versions in the active configuration.

Usage

\$\$lock_configuration(lock_mode)

Arguments

- **lock_mode**

A name that specifies the mode of the lock. Choose one of the following:

- **@read_only** — Defines the locked design objects as read-only. If you choose this lock mode, the design objects in the configuration cannot be edited by anyone, including you.
- **@write** — Defines the locked design objects as editable. If you choose this lock, you can edit the design objects contained in the configuration, but others cannot.

Return Values

VOID.

Description

Locking the active configuration enables you to perform operations on the configuration, such as building, copying, or releasing, but it prevents others from accessing the configuration object.

After you place a read-only lock on a configuration, only you can change the lock to enable editing. The read-only lock does not apply to any configuration objects that are included in the active Configuration window.

The write lock is an exclusive lock, which means after you place a write lock on a configuration, others cannot put a read-only lock or a write lock on the configuration.

To unlock your configuration, use the \$\$unlock_configuration() function.

Examples

This example opens the “config_zoo” configuration and then locks it in read-only mode.

```
$$open_configuration("$PROJECT_XYZ/config_zoo", 0, @read_write);
```

```
$$lock_configuration(@read_only);
```

Related Topics

[\\$\\$unlock_configuration\(\)](#)

[Locking a Design Configuration](#)

\$lock_configuration()

Scope: dm_config_window

Prerequisite: To use this function, you must be in a Configuration window.

Locks all design object versions in the active Configuration window.

Usage

`$lock_configuration(lock_mode)`

LOCK COnfiguration lock_mode

File > Lock Configuration

Configuration window popup menu > **Global Operations > Lock Configuration**

Arguments

- **lock_mode**

A name that specifies how this function locks the design object versions in the active Configuration window. Choose one of the following:

- **@read_only** — Defines the locked design objects as read-only. If you choose this lock mode, the design objects in the configuration cannot be edited by anyone, including you. Default.
- **@write** — Defines the locked design objects as editable. If you choose this lock mode, you can edit the design objects contained in the configuration, but others cannot.

Return Values

VOID.

Description

Locking a configuration enables you to perform operations on the configuration, such as building, copying, or releasing, but prevents others from accessing the configuration object. When you close the Configuration window, the lock on the design object versions is removed.

After you place a read-only lock on a configuration, only you can change the lock to enable editing. The read-only lock does not apply to any configuration objects that are included in the active Configuration window.

The write lock is an exclusive lock, which means after you place a write lock on a configuration, others cannot put a read-only lock or a write lock on the configuration.

You can interrupt the execution of the `$lock_configuration()` function at any time, by pressing the Kill key. When you halt the lock operation, a dialog box appears prompting you to abort or to continue the operation. If you choose to abort, the operation is halted. Design objects that have been locked prior to the interrupt remain locked; all others remain unlocked.

To check the lock status of a configuration, you use the `$report_configuration_info()` function.

Examples

1. This example locks the contents of the active Configuration window with a read-only lock.

```
$lock_configuration(@read_only);
```

2. This example locks the contents of the active configuration by using command syntax from the popup command line.

```
loc co read_only
```

Related Topics

[\\$report_configuration_info\(\)](#)

[\\$\\$unlock_configuration\(\)](#)

[Locking a Design Configuration](#)

\$\$lock_object()

Scope: dme_do_tk

Locks the specified design object.

Usage

`$$lock_object("obj_name", "type", mode, count_read_req)`

Arguments

- **obj_name**

A string that specifies the pathname of the design object.

- **type**

A string that specifies the type of the design object.

- **mode**

A name that specifies the action taken when a design object is locked.

Choose one of the following:

- **@create_version** — Increments the version number when the design object is saved.
- **@annotate** — Annotates the design object only and does not increment the version number. Default.
- **@no_modify** — Places a read-only lock on the design object.

- **count_read_req**

A Boolean that specifies whether read locks are counted. This argument only has an effect when you specify the mode option as **@no_modify**. Choose one of the following:

- **@true** — Indicates that read-only locks are counted.
- **@false** — Indicates that read-only locks are not counted. Default.

Return Values

VOID.

Description

This function enables you to enhance performance by locking a design object once and performing multiple operations on that object.

By default, this function only annotates the design object and does not increment its version number. Annotations are changes to the information about the design object, such as references and properties. No changes are made to the design object's data. To increment the version number of the design object, you must specify the **@create_version** value.

You can optionally specify that the number of requests you make for a read-only lock, on a single object, be counted. Subsequent executions of the `$$unlock_object()` function decrement the count.

When you specify the `count_read_req` argument, you must specify the mode argument as `@no_modify`. If you specify the `count_read_req` argument as `@false`, this function executes as if you had not specified the argument and subsequent requests for a `@no_modify` lock are interpreted as errors, if you already have a read-only lock on the object.

When you lock a design object with this function, you must use the `$$save_object()` function to save changes and the `$$unlock_object()` function to remove the lock.

The functions `$$lock_object()`, `$$unlock_object()`, and `$$save_object()` can be used when you perform multiple operations on a single design object. You can lock the design object, perform all your operations, saving as needed, and then unlock the object when you are finished.

The `$$lock_object()` and `$$unlock_object()` functions only operate on the current version of a design object. Operations that modify previous versions of a design object have their own internal lock and unlock routines and fail within the `$$lock_object()` and `$$unlock_object()` functions. To edit a previous version of a design object, you must use functions that can access a previous version, outside of the `$$lock_object()` and `$$unlock_object()` functions. Each function that can access a previous version locks and unlocks the design object internally as part of its operation.

Examples

1. This example locks the “sch” design object.

```
$$lock_object("$PRJ_XYZ/project/sch","Mgc_file",@create_version);
```

2. This example shows the effect of locking the object with the mode option set to `@no_modify` and the `count_read_req` argument set to `@false`.

```
$$lock_object("$PROJ_X/project/sch","Mgc_file",@nomodify, @true);
```

```
// Returns count --> 1
```

```
$$lock_object("$PROJ_X/project/sch", "Mgc_file", @nomodify, @true);  
  
//Returns count --> 2  
  
$$lock_object("$PROJ_X/project/sch", "Mgc_file", no_modify);  
  
// Error: already locked in desired mode  
  
$$unlock_object("$PROJ_X/project/sch", "Mgc_file");  
  
// count --> 1, ret = @false  
  
$$unlock_object("$PROJ_X/project/sch", "Mgc_file");  
  
// count --> 0, ret = @true  
  
$$lock_object("$PROJ_X/project/sch", "Mgc_file", no_modify);  
  
// succeeds: no count  
  
$$lock_object("$PROJ_X/project/sch", "Mgc_file", no_modify, @true);  
  
// (!) Error: already locked in desired mode  
  
// cannot mix modes
```

Related Topics

[\\$\\$save_object\(\)](#)

[\\$\\$unlock_object\(\)](#)

\$login_admin()

Scope: dmgr_proj_tk

Prerequisite: To use this function, you must not already be logged in as an administrator.

Logs the user in to administrator mode if the correct password is provided.

Usage

`$login_admin("password")`

Setup > Admin Login

Arguments

- **password**

A string that specifies the administrator password. The string value must match the current password, set by `$change_password()`, or the default password “admin1234” if the password has not been changed.

Return Values

VOID.

Examples

The following example logs in using the default password.

```
$login_admin("admin1234");
```

Related Topics

[\\$change_password\(\)](#)

[\\$logout_admin\(\)](#)

[\\$logged_in\(\)](#)

\$logged_in()

Scope: dmgr_proj_tk

Specifies whether or not the user is currently logged in as an administrator.

Usage

`$logged_in()`

Arguments

None.

Return Values

A Boolean indicating whether or not the user is currently logged in as an administrator. Returns `@true` if the user has logged in as an administrator; returns `@false` otherwise.

Examples

The following example checks for administrator mode:

```
if ($logged_in()) {  
    $message("logged in");  
}
```

Related Topics

[\\$login_admin\(\)](#)

[\\$logout_admin\(\)](#)

\$logout_admin()

Scope: dmgr_proj_tk

Prerequisite: To use this function, you must already be logged in as an administrator.

Logs the user out of administrator mode.

Usage

\$logout_admin()

Setup > Admin Logout

Arguments

None.

Return Values

VOID.

Examples

The following example logs the user out of administrator mode:

```
$logout_admin();
```

Related Topics

[\\$change_password\(\)](#)

[\\$login_admin\(\)](#)

[\\$logged_in\(\)](#)

\$maintain_hierarchy()

Sets the target path for entries in a configuration based on the specified inputs for the path prefix.

Usage

```
$maintain_hierarchy([path_prefix])
```

MAIntain HIerarchy

Arguments

- **path_prefix**

A vector of strings that is used to identify what portion of the source object's pathname is to be part of the target path for that object.

Return Values

VOID.

Description

The target path is the result of taking the configuration entry's pathname and stripping off the part of the pathname that matches the prefix. If the path_prefix does not match the beginning of the configuration entry's pathname, then the target path for that entry is not changed. The retarget paths set in this function apply to the entire configuration; no selection is necessary. Only the entries that have a path_prefix match have a retarget path set.

Related Topics

[\\$close_hierarchy\(\)](#)

[\\$\\$open_hierarchy\(\)](#)

\$\$monitor_global_status()

Scope: dm_config_tk

Reports any errors currently on the global toolkit status stack.

Usage

\$\$monitor_global_status()

Arguments

None.

Return Values

VOID.

Description

The toolkit function \$\$monitor_global_status() calls the Common User Interface error manager to report any errors that are currently on the global toolkit status stack. Error messages are sent to either the monitor window or the transcript window, depending on the monitor output setting.

Examples

This example uses the \$\$monitor_global_status() function to report errors.

```
function $move_object(destination : string)
{
    $$clear_global_status();
    local object_vector = $get_selected_objects(@full);
    overwrite = @cancel;
    $$move_object(object_vector, destination, overwrite,
        @ncreate);
    if ($$get_status_code() != 0) {
        $$monitor_global_status();
        return;
    }
}
```

Related Topics

[\\$\\$clear_global_status\(\)](#)

[\\$\\$get_status_messages\(\)](#)

[\\$\\$get_status_code\(\)](#)

[\\$\\$report_global_status\(\)](#)

[\\$\\$get_status_code_stack\(\)](#)

\$\$move_design_object()

Scope: dme_base_tk

Moves one or more design objects to a new container.

Usage

\$\$move_design_object([from], "to_string", *overwrite*, *create_dest*, *change_references*, *lock*)

Arguments

- **from**

A vector of string vectors that specifies the design objects that are to be moved. The format of this vector of string vectors is:

[[“object1”, “type1”], [“object2”, “type2”], ..., [“objectN”, “typeN”]]

The object is a string that specifies the absolute or relative pathname of the design object. The type is a string that specifies the type of the design object.

- **to_string**

A string that specifies the destination container for the design objects that are moved. The string can be a relative or an absolute pathname. If a relative pathname is specified, it is converted to an absolute pathname using the value of the MGC_WD environment variable before being stored in the reference.

- **overwrite**

A name that specifies the action taken if conflicting design objects exist in the destination container. This argument can be set to either @cancel or @replace. The default is either the last option specified or, if you have not executed this function in the current session, @cancel. The \$\$move_design_object() function *never* overwrites a container.

Choose one of the following options:

- **@cancel (-Cancel)** — If a design object exists in the destination container that has the same name as the design object being moved, the move operation is canceled and none of the specified design objects are moved to the destination container. An error message reports a target conflict and displays the name of the conflicting design object.
- **@replace (-Replace)** — When a design object exists in the destination container with the same name as a design object being moved, the conflicting design object in the destination container is replaced with the source design object if it is not a container design object. To ensure the integrity of your data, container design objects are never overwritten.

- ***create_dest***

A name that specifies the action taken when the specified destination container does not exist prior to the execution of the `$$move_design_object()` function. This argument can be set to either `@nocreate` or `@create`.

Choose one of the following options:

- **@nocreate (-NOCreate)** — If the destination container does not exist when this function executes, the move operation is not performed and an error message reports that the specified destination container does not exist. Default.
- **@create (-CReate)** — If the destination container does not exist when this function executes, the `$$move_design_object()` function attempts to create the container and to complete the move operation. If the attempt is unsuccessful, none of the specified source design objects are moved and an error message reports the reason that the destination container could not be created.

- ***change_references***

A name that specifies whether this function should attempt to update references or leave the references exactly as they were in the source design object. The default is `@change`.

Choose one of the following options:

- **@nochange (-NOCHange)** — None of the references of the design objects being moved are changed by the move operation. When the move operation is complete, all of the references of the design objects that have been moved still point to the target design object they referenced before the move operation. Default.
- **@change (-CHange)** — Any references in the design objects being moved that point to design objects that are also being moved, are changed to point to the new pathname of their target design object.

- ***lock***

A name that specifies whether the target design objects are locked before an attempt is made to change their references. The argument can be set to `@nolock` or `@lock`. The default is `@lock`. Choose one of the following options:

- **@nolock (Nolock)** — The target design objects are not locked before an attempt is made to change their references.
- **@lock (Lock)** — The target design objects are locked before an attempt is made to change their references. Default.

Return Values

An integer that indicates if the specified design objects were successfully moved. If the design objects were successfully moved, this function returns a 0 for successful completion; otherwise, this function returns a 1 for error.

Description

The iDM toolkit function `$$move_design_object()` moves all of the versions of each of the design objects specified in the “from” argument to the specified destination container. You must specify the absolute pathname and type for each design object in the from argument. This function does not update references between the design objects that are being moved, unless you specify the `change_references` option as `@change`.

This function does not allow you to move design objects that are locked or that contain locked design objects, or design objects for which you do not have write permissions.

If you specify a single source design object to move and you specify the destination using a new leaf name, this function moves the source design object to the destination container and renames the design object with the newly specified leaf name. If you specify a single source design object to move and you specify only a leaf name for the destination container, this function renames the design object in its current container.

By default, this function cancels the move operation if the specified destination container does not exist. You can specify that a destination container should be created when you move multiple design objects to a destination container that does not exist by specifying the `create_dest` argument as `@create`. However, if you specify `@create` when moving a single design object, the `@create` option does not create a new container but, instead, renames the design object to the non-existent leaf name in the current container. If any segment of your destination pathname other than the leaf name does not exist, the `@create` option has no effect and the move operation fails.

By default, the `$$move_design_object()` function cancels the move operation if conflicting design objects exist in the destination container. To replace the conflicting design objects, you choose the `@replace` value for the `overwrite` argument. When you specify `@replace`, this function overwrites any conflicting design objects, *except* for design objects that are either containers or directories.

Container type design objects are never overwritten. If you execute this function using the `@replace` option and a conflicting container design object exists in the target destination, this function moves all of the design objects specified in the from argument except for the conflicting *container* design object and displays a message stating that the container design object was not overwritten.

If you execute this function and specify the `@nolock` option, the target design objects are not locked before an attempt is made to update their references. Although specifying the `@nolock` option enables this function to execute faster, you should only use this option when you are absolutely sure that the target design objects will not be changed by another user during the function's execution. If the source design object is modified while this function is executing, your data can be corrupted.

Your ability to successfully interrupt the move operation is dependent upon what task the function is actually performing at the time you enter the interrupt. If you interrupt the move operation while the data is being moved from the source to the target destination, an attempt is made to reverse the actions of the move operation and return the data to its original state. However, if you interrupt the move operation later in the execution while the references of the

moved design objects are being updated to their new location, the changes cannot be reversed. In this case, the interrupt is canceled to prevent corruption of your data.

Examples

1. This example moves the “add_det” component from a test container to the “card_reader” container and updates its references to the new location. If an add_det component exists in the card_reader container prior to this operation, it is overwritten. If the card_reader container does not exist, this operation is not canceled because only a single design object is being moved. Instead, the add_det component is renamed to card_reader in the *\$PROJECT_XYZ* container. In this example, the move operation locks the add_det component in the target container before updating its references.

```
$move_design_object(["$PROJECT_XYZ/test4_dir/add_det",  
    "mgc_component"], "$PROJECT_XYZ/card_reader", @replace,  
    @nocreate, @change, @lock);
```

Related Topics

[\\$move_design_object\(\)](#)

\$move_design_object()

Scope: dme_base_tk

Moves one or more design objects to a new container.

Usage

`$move_design_object([from], "to_string", overwrite, create_dest, change_references, lock)`

MOVE DDesign Object [from] to_string *overwrite create_dest change_references lock*

Arguments

- **from**

A vector of string vectors that specifies the design objects that are to be moved. The format of this vector of string vectors is:

`[["object1", "type1"], ["object2", "type2"], ..., ["objectN", "typeN"]]`

The object is a string that specifies the absolute pathname of the design object. The type is a string that specifies the type of the design object.

- **to_string**

A string that specifies the destination container for the design objects that are moved. The string can be a relative or an absolute pathname. If a relative pathname is specified, it is converted to an absolute pathname using the value of the MGC_WD environment variable before being stored in the reference.

- **overwrite**

A name that specifies the action taken if conflicting design objects exist in the destination container. This argument can be set to either @cancel or @replace. The default is either the last option specified or, if you have not executed this function in the current session, @cancel. The \$move_design_object() function *never* overwrites a container.

Choose one of the following options:

- **@cancel (-Cancel)** — If a design object exists in the destination container that has the same name as the design object being moved, the move operation is canceled and none of the specified design objects are moved to the destination container. An error message reports a target conflict and displays the name of the conflicting design object.
- **@replace (-Replace)** — When a design object exists in the destination container with the same name as a design object being moved, the conflicting design object in the destination container is replaced with the source design object if the conflicting design object is not a container design object. To ensure the integrity of your data, container design objects are *never* overwritten.

- ***create_dest***

A name that specifies the action taken when the specified destination container does not exist prior to the execution of the `$move_design_object()` function. This argument can be set to either `@ncreate` or `@create`. Choose one of the following options:

- **@ncreate (-NOCReate)** — If the destination container does not exist when this function executes, the move operation is not performed and an error message reports that the specified destination container does not exist. Default.
- **@create (-CReate)** — If the destination container does not exist when this function executes, the `$move_design_object()` function attempts to create the container and to complete the move operation. If the attempt is unsuccessful, none of the specified source design objects are moved and an error message reports the reason that the destination container could not be created.

- ***change_references***

A name that specifies whether this function should attempt to update references or leave the references exactly as they were in the source design object. Choose one of the following options:

- **@nochange (-NOCHange)** — None of the references of the design objects being moved are changed by the move operation. When the move operation is complete, all of the references of the design objects that have been moved still point to the target object they referenced before the move operation.
- **@change (-CHange)** — Any references in the design objects being moved that point to design objects that are also being moved, are changed to point to the new pathname of their target design object. Default.

- ***lock***

A name that specifies whether the target design objects are locked before an attempt is made to update their references. The argument can be set to `@nolock` or `@lock`. Choose one of the following options:

- **@nolock (Nolock)** — The target design objects are not locked before an attempt is made to change their references.
- **@lock (Lock)** — The target design objects are locked before an attempt is made to change their references. Default.

Return Values

VOID.

Description

The iDM interactive function `$move_design_object()` moves all of the versions of each of the design objects specified in the “from” argument to the specified destination container. You must specify the absolute pathname and type for each design object in the from argument. This

function automatically updates the references between the design objects that are being moved, unless you specify the `change_references` option as `@nochange`.

This function does not allow you to move design objects that are locked or that contain locked objects, or design objects for which you do not have write permissions.

If you specify a single source design object to move and you specify the destination using a new leaf name, this function moves the source design object to the destination container and renames the design object with the newly specified leaf name. If you specify a single source design object to move and you specify only a leaf name for the destination container, this function renames the design object in its current container.

By default, this function cancels the move operation if the specified destination container does not exist. You can specify that a destination container should be created when you move multiple design objects to a destination container that does not exist by specifying the `create_dest` argument as `@create`. However, if you specify `@create` when moving a single design object, the `@create` option does not create a new container but, instead, renames the design object to the non-existent leaf name in the current container. If any segment of your destination pathname other than the leaf name does not exist, the `@create` option has no effect and the move operation fails.

By default, the `$move_design_object()` function cancels the move operation if conflicting design objects exist in the destination container. To replace the conflicting design objects, you choose the `@replace` value for the `overwrite` argument. When you specify `@replace`, this function overwrites any conflicting design objects that are not either containers or directories.

Container type design objects are never overwritten. If you execute this function using the `@replace` option and a conflicting container design object exists in the target destination, this function moves all of the design objects specified in the `from` argument except for the conflicting *container* design object and displays a message stating that the container design object was not overwritten.

If you execute this function and specify the `@nolock` option, the target design objects are not locked before their references are updated. Although specifying the `@nolock` option enables this function to execute faster, you should only use this option when you are absolutely sure that the target design objects will not be changed by another user during the function's execution. If the target design object is modified while this function is executing, your data can be corrupted.

When you execute the `$move_design_object()` function using either the menu interface or command syntax, the Move Design Object dialog box displays enabling you to easily enter your argument selections.

The Move Design Object dialog box contains two dialog navigators that enable you to easily select the design objects you want to move and to easily specify the destination container to which you want to move them. The dialog box also provides a filter option for each of the dialog navigators which enables you to limit the set of design objects that you see displayed in each of the dialog navigators and an **Options** button which provides you with options for specifying your overwrite, destination creation, lock, and reference update options.

Your ability to successfully interrupt the move operation is dependent upon what task the function is actually performing at the time you enter the interrupt. If you interrupt the move operation while the data is being moved from the source to the target destination, an attempt is made to reverse the actions of the move operation and return the data to its original state. However, if you interrupt the move operation later in the execution while the references of the moved design objects are being updated to their new location, the changes cannot be reversed. In this case, the interrupt is canceled to prevent corruption of your data.

Examples

1. This example moves the “8rip” component from a test container to the release library and updates its references to its new location. If a component with the same name and of the same type exists in the destination container, it is overwritten. The leaf name 8rip is added to the destination pathname and the @ncreate option is specified to ensure that the component is not moved to the parts container if the generic container does not exist. In this example, the move operation locks the 8rip component in the target container before updating its references.

```
$move_design_object( ["$DPM_PROJECT/lib_parts/test_dir2/8rip",  
    "mgc_component"], "$DPM_RLSLIB/parts/generic/8rip", @replace,  
    @ncreate, @change, @lock);
```

2. This example performs the same function as the previous example using command syntax from the popup command line.

```
mov de o["$DPM_PROJECT/lib_parts/test_dir2/8rip", "mgc_component"]  
    $DPM_RLSLIB/parts/generic/8rip -r -nocr -ch -l
```

Related Topics

[\\$\\$move_design_object\(\)](#)

\$\$move_object()

Scope: dm_config_tk

Moves the specified design objects to a target location.

Usage

\$\$move_object([object_list], “target”, *overwrite*, *create_dest*)

Arguments

- **object_list**

A vector of string vectors which specifies the design objects that are to be moved. The format of this vector of string vectors is:

[[“object1”, “type1”], [“object2”, “type2”], [“objectN”, “typeN”]]

Object is a string that specifies the absolute pathname of the design object. Type is a string that specifies the type of the design object.

- **target**

A string that specifies the destination directory of the move operation. You can specify a relative or an absolute pathname. If you specify a relative pathname, the Pyxis Project Manager application assumes the path is relative to the current working directory.

- **overwrite**

A name that specifies the action taken if conflicting design objects exist in the destination directory. Choose one of the following:

- **@cancel** — Cancels the move operation when conflicting objects are found. Default.
- **@replace** — Replaces the conflicting container type design objects in the target directory.

- **create_dest**

A name that specifies whether the destination directory is created. Choose one of the following:

- **@ncreate** — Does not create the target directory. Default.
- **@create** — Creates the target directory if it does not exist.

Return Values

VOID.

Description

The toolkit function \$\$move_object() moves the design objects specified in the “object_list” argument to the directory specified in the target argument. Design objects maintain their leaf name after the move operation is complete.

By default, this function cancels the move operation if conflicting design objects exist in the destination directory. To replace the conflicting design objects, you specify @replace as the overwrite argument.

If you execute this function on a container type object with the overwrite argument specified as @replace and a target conflict occurs, the conflicting design object is replaced as long as it is also a container type object. The replacement does not occur if either the source or the destination object are not container type objects.

If the destination directory does not exist, this function does not create the directory. To create the directory, you specify @create as the create_dest argument. If the destination directory specified in the target argument does not exist, by default, this function cancels the move operation. To create the directory and proceed with the move, you specify @create as the create_dest argument. In this case, if you specify a single design object to move and specify a destination directory that does not exist, this function renames the object with the new leaf name. If you specify multiple design objects to move and specify a destination directory that does not exist, this function creates the destination directory and copies the design objects into the directory. If you specify a destination directory in which any part of the pathname except for the leaf is non-existent, this function cancels and an error message states that the destination container does not exist.

Your ability to successfully interrupt the move operation is dependent upon what task the function is actually performing at the time you enter the interrupt. If you interrupt the move operation while the data is being moved from the source to the target destination, an attempt is made to reverse the actions of the move operation and return the data to its original state. However, if you interrupt the move operation later in the execution while the references of the moved design objects are being updated to their new location, the changes cannot be reversed. In this case, the interrupt is canceled to prevent corruption of your data.

Examples

This example moves the “plans” and “schedule” design objects in the “object_list” argument to the “project” directory.

```
$$move_object(["$PROJECT_XYZ/design/plans", "Mgc_container"],  
              ["$PROJECT_XYZ/design/schedule", "Mgc_file"],  
              "$PROJECT_XYZ/design2/project", @replace, @create);
```

Related Topics

[\\$\\$change_object_name\(\)](#)

[\\$\\$delete_object\(\)](#)

[\\$\\$copy_object\(\)](#)

\$move_object()

Scope: dmgr_model or dmgr_trash_area

Prerequisite: To use this function, you must be in an active Navigator window and must select at least one design object.

Moves the selected design object to a target location.

Usage

\$move_object(“destination”, *overwrite*)

MOVE OBJECT destination *overwrite*

Edit > Move To

Navigator window popup menu > **Edit > Move To**

Arguments

- **destination**

A string that specifies the pathname of the destination directory. You may specify either an absolute or a relative pathname.

- **overwrite**

A switch name that specifies the action taken if a conflicting design object exists in the destination directory. Choose one of the following:

- **@ask_overwrite (-Ask_overwrite)** — Asks for permission to replace the conflicting design object with the selected design object. Default.
- **@cancel (-Cancel)** — Cancels the move operation if conflicts occur.
- **@replace (-Replace)** — Replaces the conflicting design object with the selected design object.

Return Values

VOID.

Description

The interactive function \$move_object() moves the selected design object to the specified directory, maintaining the design object leaf name. This function enables you to move several design objects simultaneously. You may specify the destination directory by using either an absolute or relative pathname.

You can press the Move key, as a shortcut method for executing the \$move_object() function, as shown in “[Key Names Mapped to Functions & Scopes](#)” on page 804.

By default, this function asks for permission to replace any conflicting design objects that exist in the destination directory. To suppress the prompt and to replace the conflicting design

objects, you choose @replace as the overwrite mode argument. To cancel the move operation when conflicting design objects exist, you choose @cancel.

Moving an object by using the \$move_object() function is an interruptible operation. At any time during the execution of this function, you can press the Kill key to interrupt the operation. Your ability to successfully interrupt the move operation, however, is dependent upon what task the function is actually performing at the time you enter the interrupt.

If you interrupt the move operation while the data is being moved from the source to the target destination, an attempt is made to reverse the actions of the move operation and return the data to its original state. In some cases, the clean-up procedures might be unable to remove all the directories created by this function at the source or destination, but the source object is always undamaged. However, if you interrupt the move operation later in the execution while the references of the moved design objects are being updated to their new location, the changes cannot be reversed. In this case, the interrupt is canceled to prevent corruption of your data.

Examples

1. This example specifies an absolute pathname and the @replace value in order to move the selected design objects.

```
$move_object("$PROJECT_XYZ/d1/project", @replace);
```

2. This example moves the same design object by using command syntax from the popup command line.

```
mov ob $PROJECT_XYZ/d1/project -replace
```

3. This example specifies a relative pathname and @cancel by using command syntax from the popup command line.

```
mov ob ../ -cancel
```

Related Topics

[\\$change_object_name\(\)](#)

[\\$delete_object\(\)](#)

[\\$copy_object\(\)](#)

\$\$object_complete()

Scope: dme_do_tk

Checks whether or not the specified design object is complete.

Usage

```
$$object_complete("obj_name", "type")
```

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **type**
A string that specifies the type of the design object.

Return Values

A Boolean that indicates if the current version of the specified design object is complete. If the specified design object is complete, this function returns @true; if it is not, this function returns @false. If an error occurs, this function returns UNDEFINED.

Description

The toolkit function \$\$object_complete() checks whether or not the current version of the specified design object is complete. A design object is complete if all of its attribute and fileset members, as specified by its type definition, exist. In most cases, an object is complete if the following fileset members are present:

- An attribute file, if required by the object's type.
- Any key or required fileset members for the object.

Examples

This example checks if the "schedule" design object is complete.

```
$$object_complete("$PROJECT_XYZ/proj/doc/schedule","Mgc_file");
```

Related Topics

[\\$\\$get_fileset_members\(\)](#)

[\\$\\$object_exists\(\)](#)

\$\$object_exists()

Scope: dme_do_tk

Checks whether or not the specified design object exists.

Usage

`$$object_exists("obj_name", "type", version)`

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **type**
A string that specifies the type of the design object.
- **version**
An integer that specifies the version number of the design object. The default is the current version.

Return Values

A Boolean that indicates if the specified design object exists. If the specified design object exists, this function returns @true; otherwise, this function returns @false.

Description

For all design objects that require an attribute file, this functions checks if the attribute file exists. If the attribute file does exist, this function returns @true; if it does not, this function returns @false.

For all other designs objects, that is, objects that do not require an attribute file, this function checks if a single member of the design object's required fileset exists. If that single member is present, this function returns @true; if it is not present, this function returns @false.

By default, this function checks the current version of the specified design object. To check a previous version of a design object, you must specify the version argument.

Examples

This example checks if version 2 of the "model" design object exists.

```
$$object_exists("$PROJECT_XYZ/hardware/model", "Image_file", 2);
```

Related Topics

[\\$\\$get_fileset_members\(\)](#)

[\\$\\$object_complete\(\)](#)

\$object_status_for_rc()

Scope: dmgr_project_model

Returns the raw data used for determining the status of the specified objects.

Usage

`$object_status_for_rc([objects])`

Arguments

- **objects**

A vector containing a list of objects to execute with object status. Each object is itself a vector containing exactly the following two strings:

- The first is the object pathname.
- The second is the object type.

Return Values

VOID.

Examples

```
$object_status_for_rc([[ "$PROJECT/lib/cell", "mgc_component" ]]);
```

```
$object_status_for_rc([[ "$PROJECT/lib/cell", "mgc_component" ], [ "$PROJECT/lib/cat",  
"mgc_category" ]]);
```

Related Topics

[\\$add_project_to_rc\(\)](#)

[\\$create_work_area_from_rc\(\)](#)

[\\$cancel_checkout_objects_for_rc\(\)](#)

[\\$checkin_objects_to_rc\(\)](#)

[\\$checkout_objects_from_rc\(\)](#)

[\\$update_objects_from_rc\(\)](#)

[\\$revert_object_from_rc\(\)](#)

\$\$open_configuration()

Scope: dm_config_area, dm_config_status_area, dm_config_tk, or dm_config_window

Opens the specified configuration object.

Usage

`$$open_configuration("path", version, edit_mode)`

Arguments

- **path**
A string that specifies the pathname of the configuration object to be opened. You can specify a relative or an absolute pathname. If you specify a relative pathname, the Pyxis Project Manager application assumes the path is relative to the current working directory.
- **version**
An integer that defines the version number of the configuration object to be opened. The default is 0, which represents the current version.
- **edit_mode**
A name that specifies the mode in which the configuration is opened. Choose one of the following:
 - **@read_write** — Enables you to edit the configuration. Default.
 - **@read_only** — Does not enable you to edit the configuration.

Return Values

VOID.

Description

The toolkit function `$$open_configuration()` opens the configuration object specified by the “path” argument. When you open a configuration object, it becomes the active configuration. Remember to use the `$$save_configuration_as()` function before closing the configuration to save any changes.

To open a previous version of a configuration object, you must specify the version argument. A previous version opens in read_only mode.

Examples

1. This example opens version 4 of the “config_parts” configuration object in read_only mode.

```
$$open_configuration("$PROJECT_XYZ/config_parts",4,@read_only);
```

2. This example opens a new untitled configuration in read_write mode by specifying the path argument as an empty string.

```
$$open_configuration("");
```

Related Topics

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

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

[\\$\\$save_configuration\(\)](#)

[\\$\\$save_configuration_as\(\)](#)

\$open_configuration_window()

Scope: dmgr_session_window

Prerequisite: You can use this function from any Pyxis Project Manager window.

Opens a new or existing configuration object.

Usage

`$open_configuration_window("pathname", version)`

OPEn COnfiguration Window *pathname* *version*

Windows > Open Configuration > New | Existing

Arguments

- **pathname**
A string that specifies the absolute pathname of an existing configuration object.
- **version**
An integer that specifies the version number of the configuration that this function opens.
The default is 0, which represents the current version.

Return Values

A string that specifies the name of the Configuration window. For example, if no other Configuration windows are currently open, this function returns the string "configuration". If a Configuration window is currently open, this function returns "configuration#2".

Description

The interactive function `$open_configuration_window()` opens the configuration object specified by the `pathname` argument. To open a new untitled Configuration window, you specify the `pathname` argument as an empty string. To open a previous version of a configuration, you specify the `version` argument. You can open previous versions in read-only mode.

Examples

1. This example opens the current version of an existing configuration.

```
$open_configuration_window("$PROJ_XYZ/proj/my_config",0);
```

2. This example opens version 4 of the same configuration.

```
$open_configuration_window("$PROJ_XYZ/proj/my_config", 4);
```

3. This example opens a new untitled Configuration window.

```
$open_configuration_window("");
```

4. These examples show the command syntax from the popup command line.

```
ope co w $PROJ_XYZ/proj/my_config 0
```

```
ope co w $PROJ_XYZ/proj/my_config 4
```

```
ope co w ""
```

Related Topics

[\\$export_configuration_entries\(\)](#)

[\\$save_configuration_as\(\)](#)

[\\$save_configuration\(\)](#)

[Configuration Objects](#)

\$\$open_hierarchy()

Scope: dmgr_project_model

Opens the specified hierarchy (project, external library, or technology library) in the Project Navigator window.

Usage

\$\$open_hierarchy("path")

opehi

OPEn Hierarchy

File > Open > Hierarchy

Arguments

- **path**

A string that specifies the path of the hierarchy to be opened.

This may be a hard path or soft path based on either an environment variable or an entry in the active location map at the time the function is called.

Return Values

VOID.

Examples

The following example opens a project at */tmp/project*:

```
$$open_hierarchy("/tmp/project");
```

Related Topics

[\\$close_hierarchy\(\)](#)

[\\$maintain_hierarchy\(\)](#)

\$open_navigator()

Scope: dmgr_session_window

Prerequisite: You can use this function from any Pyxis Project Manager window.

Opens the Navigator window.

Usage

`$open_navigator("pathname", mode)`

OPEn NAvigator *pathname mode*

Windows > Open Navigator > View by Icon | View by Name

Arguments

- ***pathname***
A string that specifies the absolute pathname where the Navigator window is rooted. The default is the current working directory.
- ***mode***
A switch name that specifies whether the Navigator window displays design objects in iconic or list mode.
 - **@iconic (-Iconic)** — Opens an iconic navigator. Default.
 - **@list (-List)** — Opens a list navigator.

Return Values

A string that specifies the name of the navigator. For example, if no other navigators are currently open, this function returns the string "navigator". If a Navigator window is currently open, this function returns "navigator#2".

Description

The interactive function `$open_navigator()` opens a Navigator window. By default, this function opens an iconic navigator rooted at the working directory. To open a list navigator, you specify `@list` as the mode argument. To open a Navigator window rooted at a directory other than the MGC working directory, you specify an absolute pathname that is a pathname to a directory.

You can press the Navigator key, as a shortcut method for opening an iconic navigator, as shown in "[Key Names Mapped to Functions & Scopes](#)" on page 804.

Examples

1. This example opens a list navigator rooted at the `$PROJECT_XYZ/project` directory.

```
$open_navigator("$PROJECT_XYZ/project", @list);
```

2. This example opens the same navigator by using command syntax on the popup command line.

```
ope na $PROJECT_XYZ/project -list
```

Related Topics

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

[\\$view_by_name\(\)](#)

[\\$view_by_icon\(\)](#)

\$open_object()

Scope: dmgr_model or dmgr_version_model

Prerequisite: To use this function, you must be in an active Navigator window and must select exactly one design object.

Opens the selected design object.

Usage

`$open_object("tool_name")`

OPEn OBject *tool_name*

File > Open

Navigator or **Project Navigator** window > **Open**

Argument

- *tool_name*

A string that specifies the name of the tool to be invoked when the design object is opened.

Return Values

VOID.

Description

This function invokes either the default tool or the tool specified by the *tool_name* argument. The Pyxis Project Manager application executes the appropriate qualification script.

You can also use the `OpenObject` key as a shortcut method for executing the `$open_object()` function on the selected design object as shown in “[Key Names Mapped to Functions & Scopes](#)” on page 804.

The tool is responsible for dealing with any unique requirements for opening any version, other than the current version, of the selected object. If this is not possible, version selection must be done in the context of the tool, not from the Pyxis Project Manager application.

Examples

1. This example opens the selected design object.

```
$open_object("Editor");
```

2. This example opens the selected design object and specifies which tool to invoke by using command syntax from the popup command line.

```
ope ob Editor
```

3. This example invokes the selected design object's default tool by using command syntax from the popup command line.

ope ob

Related Topics

[\\$open_tool\(\)](#)

\$open_read_only_editor()

Scope: dmgr_model

Prerequisite: To use this function, you must be in an active Navigator window and must select exactly one ASCII file.

Invokes the read-only ASCII editor on the selected object.

Usage

\$open_read_only_editor()

Open > Read-Only Editor

Arguments

None.

Return Values

VOID.

Description

The interactive function \$open_read_only_editor() opens the specified read-only editor on a selected ASCII file. If no startup files exist, the default read-only editor is the Notepad. If a startup file exists that specifies an editor other than the Notepad, or if you have specified a different default editor using the function \$setup_default_editor(), your specified editor opens the file.

If you execute the \$open_read_only_editor() function with multiple objects selected, a dialog navigator displays enabling you to select the object you want to open. If you execute this function on an object of the type “Mgc_file”, the read-only editor is invoked on that object.

When you execute this function on a single object that is not of the type “Mgc_file”, the default tool for that object's type is invoked. If the object's default tool is “Editor”, the read-only editor is invoked on the object.

If you execute the \$open_read_only_editor() function on a directory, the contents of the directory are explored. If you execute the function on a configuration object, the Configuration window opens in edit mode. If you execute this function on an object whose type does not have a default tool defined, an error message states that no default tool exists for this type.

Examples

This example opens your specified read-only editor on the selected ASCII file.

```
$open_read_only_editor();
```

Related Topics

[\\$setup_default_editor\(\)](#)

\$open_session_monitor()

Scope: dmgr_session_window

Makes the Session Monitor window visible.

Usage

`$open_session_monitor()`

OPEn SEssion Monitor

Windows > Open Session Monitor

Arguments

None.

Return Values

VOID.

Description

The interactive function `$open_session_monitor()` opens the session monitor window. The session monitor is automatically cleared upon each new configuration management toolkit operation.

You can customize your session setup values to specify that the session monitor window is either visible or hidden during session configuration operations.

You can press the Monitor key, as a shortcut method of executing this function, as shown in “[Key Names Mapped to Functions & Scopes](#)” on page 804.

Examples

1. This example displays the session monitor window.

```
$open_session_monitor();
```

2. This example displays the session monitor window, by using command syntax from the popup command line.

```
ope se m
```

Related Topics

[\\$\\$clear_monitor\(\)](#)

[\\$show_monitor\(\)](#)

[\\$hide_monitor\(\)](#)

[\\$\\$writeln_monitor\(\)](#)

[Change Your Session Setup](#)

\$\$open_tool()

Scope: dm_tool_tk

Invokes the specified tool.

Usage

```
$$open_tool("tool_name", "tool_type", "data_object_name", "data_object_type",  
            data_object_ver)
```

Arguments

- **tool_name**
A string that specifies the name of the tool to be invoked. The string can be either the full pathname of the tool or the leaf name. If you enter a leaf name, the toolbox search path is used to resolve the string.
- **tool_type**
A string that specifies the type of the tool to be invoked. The default is the empty string "".
- **data_object_name**
A string that specifies the data object to be used as input for the tool invocation. The default is the empty string "".
- **data_object_type**
A string that specifies the type of the data object that is used as input for the tool. The default is the empty string "".
- **data_object_ver**
An integer that specifies the correct version of the data object to be used as input for the tool. The default is zero, which specifies the current version.

Return Values

VOID.

Description

The Pyxis Project Manager application executes the qualification script associated with the tool. If a data object is specified in the optional data_object argument, the tool invokes using that data object as input.

If a full pathname is specified for the tool name, a tool type must be specified in the tool_type argument. The tool type may or may not be available in the Tools window, but it must be a valid tool viewpoint type. If a leaf name is specified for the tool name, the tool type may or may not be specified. If a tool type is specified, it is used.

Examples

The following example opens the specified tool “Editor” on the Notepad file “junk”.

```
$$open_tool("Editor","Mgc_default_ascii_editor","junk",  
            "Mgc_file",0);
```

Related Topics

[\\$open_tool\(\)](#)

\$open_tool()

Scope: dmgr_tool_window

Prerequisite: To use this function, you must be in an active Tools window and must select exactly one tool.

Invokes the selected tool.

Usage

\$open_tool()

OPEn TOol

File > Open

Arguments

None.

Return Values

VOID.

Description

The Pyxis Project Manager application executes the qualification script associated with the tool. In this type of invocation, the tool is responsible for selecting or creating required input data that it may need.

You can press the OpenObject key, as a shortcut method for executing the \$open_tool() function, as shown in “[Key Names Mapped to Functions & Scopes](#)” on page 804.

Examples

1. This example opens the selected tool.

```
$open_tool();
```

2. This example opens the selected tool by using command syntax from the popup command line.

```
ope to
```

Related Topics

[\\$open_object\(\)](#)

[Tool Invocation](#)

\$open_tools_window()

Scope: dmgr_session_window

Prerequisite: You can use this function from any Pyxis Project Manager window.

Opens a Tools window.

Usage

`$open_tools_window()`

OPEn TOols Window

Windows > Open Tools Window

Arguments

None.

Return Values

A string that specifies the name of the Tools window. If no other Tools windows are open, this function returns the string “tools”. If a Tools window is currently open, this function returns “tools#2”.

Description

The Tools window displays all of the tools available in the current session. You can change this display by changing the toolbox search path, which dictates the contents of the Tools window.

Examples

1. This example opens the Tools window.

`$open_tools_window();`

2. This example opens a Tools window by using command syntax from the popup command line.

ope to w

Related Topics

[\\$add_toolbox\(\)](#)

[\\$view_toolboxes\(\)](#)

[\\$get_toolbox_search_path\(\)](#)

[\\$view_tools\(\)](#)

[\\$remove_toolbox\(\)](#)

\$open_trash_window()

Scope: dmgr_session_window

Prerequisite: You can use this function from any Pyxis Project Manager window.

Opens a Trash window, which displays the contents of the trash can.

Usage

\$open_trash_window()

OPEn TRash Window

Windows > Open Trash Window

Arguments

None.

Return Values

A string that specifies the name of the Trash window. If no other Trash windows are open, this function returns the string “trash”. If a Trash window is currently open, it returns UNDEFINED.

Description

You can delete design objects by dragging them to the trash can or a Trash window or by using the \$trash_object() function. You can remove design objects by dragging them out of the Trash window or by using the \$untrash_object() function. Design objects in the Trash window are not permanently deleted until you empty the Trash window by executing the \$empty_trash() function.

Only one Trash window can be opened at a time. If you try to open more than one Trash window, this function pops the opened Trash window to the foreground.

Examples

1. This example opens the Trash window.

```
$open_trash_window();
```

2. This example opens the Trash window by using command syntax from the popup command line.

```
ope tr w
```

Related Topics

[\\$delete_object\(\)](#)

[\\$trash_object\(\)](#)

[\\$untrash_object\(\)](#)

[\\$empty_trash\(\)](#)

\$open_types_window()

Scope: dmgr_session_window

Prerequisite: You can use this function from any Pyxis Project Manager window.

Displays the types currently loaded in the Pyxis Project Manager application.

Usage

`$open_types_window("title")`

OPEn TYpes Window *"title"*

Windows > Open Types Window

Arguments

- *title*

A string that specifies the types window to be displayed. The default is the string "Type Manager: Known Type Reps".

Return Values

A string that specifies the name of the types window. For example, if no other types windows are open, this function returns the string "type". If a type window is currently open, this function returns "type#2"

Description

The interactive function `$open_types_window()` brings up a read-only window that displays a sorted list of the design object types currently loaded in the Pyxis Project Manager application. This list does not contain any duplicate entries. If more than one occurrence of a type exists, only the most recently added type is listed.

From the types window, you can load new type registries and display the properties of each type. The Pyxis Project Manager application also provides options to search the types window, both forward and backward for a particular entry, and to unselect all objects in the window.

Examples

1. This example brings up the types window.

```
$open_types_window();
```

2. This example displays the types window by using command syntax from the popup command line.

```
open ty w
```

Related Topics

[\\$load_registry\(\)](#)

[\\$report_type_info\(\)](#)

\$\$open_versioned_object()

Scope: dme_do_tk

Opens a versioned design object that encapsulates data from a non-Mentor Graphics application.

Usage

```
$$open_versioned_object("obj_name", "obj_type")
```

Arguments

- **obj_name**
A string that specifies the pathname of the versioned design object.
- **obj_type**
A string that specifies the type of the versioned design object.

Return Values

A Boolean that indicates if the versioned design object was opened. If the design object opens successfully, this function returns @true. Otherwise, it returns @false.

Description

The toolkit function \$\$open_versioned_object() opens a working copy of a versioned design object that encapsulates a non-Mentor Graphics data file.

You should use this function in conjunction with the \$\$close_versioned_object() and \$\$create_versioned_object() functions. This set of functions enable you to apply the Mentor Graphics version management scheme to design objects that encapsulate data from non-Mentor Graphics applications.

Note



Note that the use of the \$\$close_versioned_object(), \$\$create_versioned_object(), and \$\$open_versioned_object() functions is not always required for encapsulating data from non-Mentor Graphics applications. The only situation where these functions are required is for managing *versions* of data from non-Mentor Graphics applications. For information about encapsulating data, refer to the *Pyxis Registrar User's and Reference Manual* on SupportNet.

The \$\$open_versioned_object() function operates on design objects created by the \$\$create_versioned_object() function. When you open a versioned design object, \$\$open_versioned_object() places a lock on the design object, then opens a working copy of the current version. You can use the appropriate non-Mentor Graphics application(s) make changes to the working copy.

To close the design object, you must use the `$$close_versioned_object()` function which removes the lock from the object, saves the data, and updates the attributes and version information about the object.

Examples

Assume that you have a third-party tool that works on files with the suffix *.hck*, and that the *Phantom* type has a *.hck* fileset member. You have registered the *Phantom* type and the third-party tool in your local registry.

This example opens the *my_file* design object of *Phantom* type.

```
$$open_versioned_object("$PROJECT_XYZ/my_file", "Phantom");
```

At this point, the following files exist:

```
my_file.Pantom.attr
```

```
my_file.Pantom.lck
```

```
my_file.hck_1 //Versions present prior to
```

```
my_file.hck_2 //calling this function.
```

```
my_file.hck // The working copy.
```

Related Topics

[\\$\\$close_versioned_object\(\)](#)

[\\$\\$create_versioned_object\(\)](#)

\$\$prune_design_hierarchy()

Scope: dm_config_tk

Prunes off back versions of a design object.

Usage

\$\$prune_design_hierarchy([object_list], version_depth, *prune_mode*)

Arguments

- **object_list**

A vector of string vectors that specifies the design objects to be pruned. The format of the vector of string vectors is:

```
[["name1", "type1", version], ["name2", "type2", version], ...,  
 ["nameN", "typeN", version]]
```

The name is a string that specifies the absolute pathname of the design object. The type is a string that specifies the type of the design object. Version is an optional integer that specifies the version of the design object.

- **version_depth**

An integer that specifies the new version depth of the specified design object.

- ***prune_mode***

A switch name that specifies whether the version depth changes should be applied to those design objects that exist within the specified design object's containment or reference hierarchy. By default, the mode is @containment. Choose one of the following:

- **@containment** — Changes the version depth of the specified design object and all of the objects contained within the specified design object. Default.
- **@reference** — Changes the version depth of the specified design object and all of the objects contained within or referenced by either the specified design object or by any objects contained within the specified design object.

Return Values

VOID.

Description

The toolkit function \$\$prune_design_hierarchy() changes the version depth of the design objects specified in the object_list argument and prunes off (deletes) all previous versions of those design objects that exceed the new depth specified by the version_depth argument. This function does not delete frozen versions or versions that are latched by the Pyxis Design Viewpoint application.

Specifying the `version_depth` argument as 0 sets the depth to the default version depth, as specified by the design object's type. To prevent deleting previous versions, you set the version depth to -1, which enables the design object to have an infinite number of previous versions.

By default, this function changes the version depth of the specified design objects and of all of the versioned design objects contained within the specified design objects. If you specify the `prune_mode` argument as `@reference`, this function changes the version depth of the specified design objects and of all of the versioned design objects contained within or referenced by the specified design objects or by any object contained within them.

You can specify that this function prunes a particular version of a design object by using a version specifier in the `object_list` argument.

Examples

This example sets the version depth of the design object `new_config` to the default version depth for design objects of the type *Dme_config_do*, and deletes all of its versions and all of the version of any design objects it contains except for the current version, the previous version, and any frozen or latched versions.

```
$$prune_design_hierarchy(["$PROJECT_XYZ/new_config",  
    "Dme_config_do"], 0, @containment);
```

\$\$read_map()

Scope: dme_dn_tk

Reads the specified ASCII location map file into memory.

Usage

```
$$read_map("file_name")
```

Arguments

- **file_name**

A string that specifies the absolute pathname of the location map to be read.

Return Values

VOID.

Description

The maximum number of entries that can be read into memory is 2048. When the number of entries exceeds this limit, an error message is reported.

If you enter an empty string or if a location map does not exist at the location specified by the pathname, one of two events occur:

- If a location map has been read into memory during the current session, that map is read again.
- If a location map has *not* been read, the value of the MGC_LOCATION_MAP environment variable is searched.

If the value of MGC_LOCATION_MAP is set to "NO_MAP", the assumption is made that a location map is not used. If the MGC_LOCATION_MAP environment variable is not set, the location map is searched for at the following locations in the following order:

- ./mgc_location_map*
- \$HOME/mgc/mgc_location_map*
- \$MGC_HOME/etc/cust/mgc_location_map*
- \$MGC_HOME/shared/etc/cust/mgc_location_map*

If a location map is not found in any of these locations, an error message is returned stating that your location map cannot be located. If a location map already exists in memory, it is deleted and replaced with the new map. The values of existing environment variables override hard pathname entries in the location map.

When \$\$read_map() encounters an error in reading the location map, the function ignores the map and sends an error message to the transcript. In this case, you should call the function \$\$handle_map_error() to let the user specify how to proceed.

Examples

This example reads the private location map *\$PROJECT_XYZ/project/d1/mgc_location_map* into memory.

```
$$read_map("$PROJECT_XYZ/project/d1/mgc_location_map");
```

Related Topics

[\\$change_location_map_entry\(\)](#)

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

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

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

[\\$\\$handle_map_error\(\)](#)

[\\$\\$show_location_map\(\)](#)

\$read_map()

Scope: session_area

Prerequisite: You can use this function from any Pyxis Project Manager window.

Reads the specified ASCII location map file into memory.

Usage

`$read_map("file_name")`

REAd Map file_name

MGC > Location Map > Read Map

Arguments

- **file_name**

A string that specifies the absolute pathname of the location map to be read.

Return Values

VOID.

Description

The maximum number of entries that can be read into memory is 2048. When the number of entries exceeds this limit, an error message is reported.

If you enter an empty string or if a location map does not exist at the location specified by the pathname, one of two events occur:

- If a location map has been read into memory during the current session, that map is read again.
- If a location map has *not* been read, the value of the MGC_LOCATION_MAP environment variable is searched.

If the value of MGC_LOCATION_MAP is set to "NO_MAP", the assumption is made that a location map is not used. If the MGC_LOCATION_MAP environment variable is not set, the location map is searched for at the following locations in the following order:

- ./mgc_location_map*
- \$HOME/mgc/mgc_location_map*
- \$MGC_HOME/etc/cust/mgc_location_map*
- \$MGC_HOME/shared/etc/cust/mgc_location_map*

If a location map is not found in any of these locations, an error message is returned stating that your location map cannot be located. If a location map already exists in memory, it is deleted and replaced with the new map. The values of existing environment variables override hard pathname entries in the location map.

When `$read_map()` encounters an error in reading the location map, the function ignores the map, sends an error message to the transcript, and calls the function `$$handle_map_error()`. The `$$handle_map_error()` function presents the user with a dialog box to let the user specify how to proceed.

Examples

This example reads the private location map `$PROJECT_XYZ/project/d1/mgc_location_map` into memory.

```
$read_map("$PROJECT_XYZ/project/d1/mgc_location_map");
```

Related Topics

[\\$change_location_map_entry\(\)](#)

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

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

[\\$show_location_map\(\)](#)

[\\$\\$handle_map_error\(\)](#)

\$refresh_all()

Scope: dmgr_project_model

Resynchronizes the objects displayed within the Project Navigator window with the objects that exist on disk.

Usage

\$refresh_all()

REFresh ALl

Windows > Refresh All

Arguments

None.

Return Values

VOID.

Examples

\$refresh_all()

Related Topics

[\\$set_project_refresh_heartbeat\(\)](#)

\$\$release_configuration()

Scope: dm_config_tk

Releases all design object versions in the active configuration to a target location.

Usage

\$\$release_configuration("target", "comments", create_dest, update_refs)

Arguments

- **target**

A string that defines the destination directory of the release. You can specify a relative or an absolute pathname. If you specify a relative pathname, the Pyxis Project Manager application assumes the path is relative to the current working directory.

- **comments**

A string that specifies the release annotation. The default is "", which is an empty string.

- **create_dest**

A name that specifies the action taken if the destination directory does not exist. Choose one of the following:

- **@ncreate** — Does not create the destination directory and cancels the release operation. Default.
- **@create** — Creates the destination directory and proceeds with the release operation.

- **update_refs**

A name that specifies whether or not to update references on the released configuration. Choose one of the following:

- **@noupdate** — Does not update the references on the released configuration.
- **@update** — Updates the references on the released configuration. Default.

Return Values

VOID.

Description

The toolkit function \$\$release_configuration() creates, in the specified destination directory, protected copies of the design object versions in the active configuration.

If you have saved the configuration and are using a location map, in addition to releasing the design object versions and the configuration object, this function also releases the in-memory location map to the destination directory. The name of the location map object is of the format:

configuration_name.cfg_map_cfg_version_num

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

If you have not saved the configuration, this function does not release a configuration object or the location map to the destination directory.

If conflicting unversioned design objects exist in the destination directory, this function replaces the unversioned design objects. If you are releasing versioned design objects, this function creates a new version of the object rather than replace the existing one.

If the destination directory specified in the `target` argument does not exist, by default, this function cancels the release operation. To create the directory and proceed with the release, you specify `@create` as the `create_dest` argument.

This function updates the references held by all objects in the release, to reflect the new location.

This function stores the annotation string as a version property. To view the annotation string, use the design object toolkit function `$$get_version_properties()`.

Examples

This example opens the `config_a` configuration, then releases it to the `$PROJECT_XYZ/project/misc` target directory, and updates its references. The release is annotated with the “`test_release`” string. If the directory does not exist, this function creates it.

```
$$open_configuration("$PROJECT_XYZ/d1/config_a", 0, @read_write);  
  
$$release_configuration("$PROJECT_XYZ/project/misc", "test_release",  
    @create, @update);
```

Related Topics

[\\$\\$copy_configuration\(\)](#)

[\\$\\$get_version_properties\(\)](#)

[Configuration Management Concepts](#)

\$release_configuration()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window.

Releases the design object versions in the active Configuration window to a target location.

Usage

`$release_configuration("destination_dir", "comment", preview_mode, update_refs)`

RELease COntfiguration destination_dir comment preview_mode update_refs

Edit > Release Configuration

Configuration window popup menu > **Global Operations** > **Release**

Arguments

- **destination_dir**
A string that specifies the absolute pathname of the destination directory.
- **comment**
A string that annotates any comments to be appended to the release. The annotation string is stored as a version property. The default is "", which is an empty string.
- **preview_mode**
A switch name that specifies whether this function issues a preview report. Choose one of the following:
 - **@nopreview (-Nopreview)** — Does not issue the preview report; instead, releases the configuration. Default.
 - **@preview (-Preview)** — Issues the preview report only and does not release the configuration.
- **update_refs**
A switch name that specifies whether or not to update references on the released configuration. Choose one of the following:
 - **@nouupdate (-NOUpdate)** — Does not update the references on the released configuration.
 - **@update (-Update)** — Updates the references on the released configuration. Default.

Return Values

VOID.

Description

The interactive function `$release_configuration()` creates, in the specified destination directory, the design object versions in the active Configuration window.

If you have saved the configuration and are using a location map, in addition to releasing the design object versions and the configuration object, this function also releases the in-memory location map to the destination directory. The name of the location map object is of the format:

`configuration_name.cfg_map_cfg_version_num`

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

If you have not saved the configuration, this function does not create a configuration object or release the location map to the destination directory.

If the destination directory contains unversioned design objects with the same name as those in the configuration, this function replaces the unversioned design objects. If you are releasing versioned design objects, this function creates a new version of the object rather than replace the existing one.

If you set the target path for a retargetable entry, that entry is released to the location specified in the target path and not to the destination directory specified for the release operation.

If you specify a destination directory that does not exist, this function brings up a dialog box that asks for permission to create the directory. If the parent of the specified directory does not exist, this function returns an error message and cancels the release.

The annotation string enables you to annotate the release. The annotation string is stored as a version property. To view the annotation, you navigate to the directory that you specified as the destination directory for the release. You then select the appropriate configuration object and execute the **Report > Show Versions** menu item. Next, you select the correct version from the version window. Finally, you execute the **Report Version Info** popup menu item.

The preview argument enables you to view the release prior to the actual release operation. To issue the preview report, you specify `@preview` as the preview argument.

The `$release_configuration()` function updates the references held by all objects in the release to reflect the new location.

The `$release_configuration()` function writes a status report to the configuration's monitor window, as it executes. The report includes the number of failures, errors, and warnings found during the operation. Additionally, a message is written to the message area stating whether the release operation failed, passed, or passed with warnings.

If your session setup values specify to show the configuration monitor window, the monitor window is opened when you execute this function. When the operation is complete, the monitor window remains visible until you return to the Configuration window by executing the monitor window's popup menu item **Hide Monitor**. You can redisplay the operation's status report after the operation, by executing the Configuration window's popup menu item **Show Monitor**. For

information about specifying your session setup values, refer to “[Changing Your Session Setup](#)” in the *Pyxis Project Manager User's Manual*.

The configuration's monitor window is cleared upon the execution of a new configuration operation. To save the status information in a file, you can use the monitor window's popup menu item **Export**.

Releasing a configuration by using the `$release_configuration()` function is an interruptible operation. At any time during the execution of this function, you can press the Kill key to interrupt the operation. When you halt the release operation, a dialog box appears prompting you to abort or to continue the operation. If you choose to abort, the operation is halted.

Interrupting the release operation in this way, halts the operation and attempts to return the configuration to its original state by deleting the directories created by the function. However, these clean-up procedures do not remove all of the directories that this function creates at the source or destination. In order to ensure that containers which have only one version are not destroyed, only containers that are completely empty are deleted.

Examples

1. This example releases the contents of the active Configuration window to the `$PROJ_XYZ/proj/misc` directory, updates the references of the design objects contained within the configuration, and annotates the release with the “test_release” string.

```
$release_configuration("$PROJ_XYZ/proj/misc", "test_release", @nopriview,  
@update);
```

The following messages are reported in the configuration monitor window

```
Releasing Configuration...  
Checking for Conflicts...  
Conflict Resolution Complete  
Processing Entries...  
Entry Processing Complete  
Updating References...  
Reference Update Complete  
Release Complete
```

2. This example releases the same configuration by using command syntax from the popup command line.

```
rel co $PROJ_XYZ/proj/misc test_release -nopriview
```

Related Topics

[\\$build_configuration\(\)](#)

[\\$set_target_path\(\)](#)

[\\$copy_configuration\(\)](#)

[Releasing a Design Configuration](#)

[Change Your Session Setup](#)

\$\$release_object()

Scope: dm_config_tk

Releases the current version of the specified objects to the specified destination directory.

Usage

`$$release_object([object_list], "target", "release_name", comment, [filter_list], follow_refs, preview_mode, create_dest, update_refs)`

Arguments

- **object_list**

A vector of string vectors that specifies which design objects that are to be released. The format of this vector of string vectors is:

`[["object1", "type1"], ["object2", "type2"], ["objectN", "typeN"]]`

Object is a string that specifies the absolute pathname of the design object. Type is a string that specifies the type of the design object.

- **target**

A string that specifies the destination directory of the release operation. You can specify a relative or an absolute pathname. If you specify a relative pathname, the Pyxis Project Manager application assumes the path is relative to the current navigator directory.

- **release_name**

The name of the released configuration in the destination directory. The default is "release_config".

- ***comment***

A string that annotates any comments to be appended to the release. The annotation string is stored as a version property. The default is "", which is an empty string.

- ***filter_list***

A vector of string vectors that specifies which libraries parts are to be excluded from the release, based on their pathname. Filtering is based on string matching. If you specify a hard pathname, the pathname is not converted to its soft pathname equivalent. You can specify the pathname pattern by entering the literal string or by using UNIX System V regular expressions. The default is to include all libraries in the release.

- ***follow_refs***

A name that enables you to set reference traversal to either on or off. By default, containment traversal is always set to on, which means that a design object's contained objects are added to the configuration during a build. Choose one of the following:

@follow — All externally referenced design objects of the selected design objects, as well as internally referenced design objects, are included in the released object during the release.

@nofollow — Externally referenced design objects of the selected design objects, are not included in the released object during the release. The default setting for reference traversal is off.

- ***preview_mode***

A switch name that specifies whether this function issues a preview report. Choose one of the following:

- **@nopreview** — Does not issue the preview report; instead, releases the configuration. Default.
- **@preview** — Issues the preview report only and does not release the configuration.

- ***create_dest***

A name that specifies whether the destination directory is created.

Choose one of the following:

- **@ncreate** — Does not create the target directory. Default.
- **@create** — Creates the target directory if it does not exist.

- ***update_refs***

A name that specifies whether or not to update references on the released object.

Choose one of the following:

- **@noupdate** — Does not update the references on the released object.
- **@update** — Updates the references on the released object. Default.

Return Values

VOID.

Description

The toolkit function `$$release_object()` releases the current version of the specified objects, as well as all objects they contain, to the specified destination directory. Any object that is included in the release as a result of its containment or reference relationship to one of the specified objects is released as a current version, unless it is referenced as a fixed version.

If a location map is being used, in addition to releasing the design object versions and the configuration object, this function also releases the in-memory location map to the destination directory. The name of the location map object is of the format:

`configuration_name.cfg_map_cfg_version_num`

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

Any objects that are related to the specified objects by reference can be excluded from the release operation by setting reference traversal to off. If you want to include referenced objects

in the release, but exclude externally referenced libraries, you can set `reference` to on and use the `filter` argument to filter out unwanted libraries based on pattern matching.

By default, all libraries are included in the release. When you specify a hard pathname to exclude libraries from the release, the pathname is treated as a string and the pathname is not resolved to its soft path equivalent.

Unrecognized objects, contained by the objects being released, are released to the target destination; recognized corrupted objects are not released. You can specify that error reports about unrecognized objects, corrupted objects, broken references, dangling references, and other release information are reported.

If the destination directory specified in the `target` argument does not exist, by default, this function cancels the release operation. To create the directory and proceed with the release, you specify `@create` as the `create_dest` argument. In this case, if you specify a single design object to release and specify a destination directory that does not exist, this function renames the object with the new leaf name. If you specify multiple design objects to release and specify a destination directory that does not exist, this function creates the destination directory and copies the design objects into the directory. If you specify a destination directory in which any part of the pathname except for the leaf is non-existent, this function cancels an error message that states that the destination container does not exist.

You use the `@preview` option to preview the results of the release operation prior to actually executing it. When you specify the `@preview` option, you view the preview report in the session configuration monitor window.

The status of the release operation is monitored in real-time. Information about the release, whether it results from the normal monitoring process that occurs or from setting the `@preview` option is reported to the session configuration monitor window. You can view the session configuration monitor window by executing the session window pulldown menu item

Windows > Open Session Monitor. If your session setup values specify to show the session monitor window, the monitor window is automatically opened when you execute this function.

If you want to view the status reports of your toolkit operations in the transcript window, you can change the Pyxis Project Manager application defaults by executing the interactive function `$setup_monitor()` from the popup command line or via its corresponding menu item **Setup > Monitor Defaults**. For complete information about specifying your session setup values, refer to [“Changing Your Session Setup”](#) in the *Pyxis Project Manager User's Manual*.

Examples

This example opens the *object_config* design object, then releases it to the *\$PROJ_DESIGN/design1/misc* target directory, and updates its references. The release is annotated with the “my_test_release” string. Both reference traversal and the preview switch are set to off. If the directory does not exist, this function creates it.

```
$$open_configuration("$PROJ_DESIGN/design1/object_config",  
0, @read_write);
```

```
$$release_object(["$PROJ_DESIGN/design1j/object_config",  
  "Mgc_file"], "$PROJ_DESIGN/design1j/misc", "release_config",  
  "my_test_release", [], @nofollow, @nopreview, @create, @update);
```

Related Topics

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

[\\$\\$copy_configuration\(\)](#)

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

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

[\\$open_session_monitor\(\)](#)

[\\$setup_monitor\(\)](#)

\$release_object()

Scope: dmgr_model

Prerequisite: To use this function, you must be in an active Navigator window and must select at least one object.

Releases the current version of the selected objects to the specified destination directory.

Usage

`$release_object("target", "rls_name", "rls_comment", [filter], follow_refs, preview_mode, update_refs)`

RELEase OBject target *rls_name* *rls_comment* *filter* *follow_refs* *preview_mode* *update_refs*

File > Release Design

Configuration window popup menu > **Release Design**

Arguments

- **target**
A string that specifies the destination directory of the release operation. You can specify a relative or an absolute pathname. If you specify a relative pathname, the Pyxis Project Manager application assumes the path is relative to the current navigator directory.
- **rls_name**
A string that specifies the name of the configuration object in the destination directory. The default is "release_config".
- **rls_comment**
A string that annotates any comments to be appended to the release. The annotation string is stored as a version property. The default is "", which is an empty string.
- **filter**
A vector of string vectors that specifies which libraries parts are to be excluded from the release, based on their pathname. Filtering is based on string matching. If you specify a hard pathname, the pathname is not converted to its soft pathname equivalent. You can specify the pathname pattern by entering the literal string or by using UNIX System V regular expressions. The default is to include all libraries in the release.
- **follow_refs**
A name that enables you to set reference traversal to either on or off. By default, containment traversal is always set to on, which means that a design object's contained objects are added to the configuration during a build. Choose one of the following:
 - **@follow (-Follow)** — All externally referenced design objects of the selected design objects, as well as internally referenced design objects, are included in the released object during the release.

- **@nofollow (-NOFollow)** — Externally referenced design objects of the selected design objects, are not included in the released object during the release. Default.
- ***preview_mode***
 A switch name that specifies whether this function issues a preview report. Choose one of the following:
 - **@nopreview (-Nopreview)** — Does not issue the preview report; instead, releases the configuration. Default.
 - **@preview (-Preview)** — Issues the preview report only and does not release the configuration.
- ***update_refs***
 A switch name that specifies whether or not to update references on the released object. Choose one of the following:
 - **@noupdate (-NOUpdate)** — Does not update the references on the released object. Default.
 - **@update (-Update)** — Updates the references on the released object.

Return Values

VOID.

Description

The interactive function `$release_object()` releases the current version of the specified objects, as well as all objects they contain to the specified destination directory. Any object that is included in the release, as a result of its containment or reference relationship to one of the specified objects, is released as a current version, unless it is referenced as a fixed version.

If a location map is being used, in addition to releasing the design object versions and the configuration object, this function also releases the in-memory location map to the destination directory. The name of the location map object is of the format:

`configuration_name.cfg_map_cfg_version_num`

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

Any objects that are related to the specified objects by reference can be excluded from the release operation by setting reference traversal to off. If you want to include referenced objects in the release, but exclude externally referenced libraries, you can set reference to on and use the filter argument to filter out unwanted libraries based on pattern matching. By default, all libraries are included in the release. When you specify a hard pathname to exclude libraries from the release, the pathname is treated as a string and the pathname is not resolved to its soft path equivalent.

Unrecognized objects, contained by the objects being released, are released to the target destination; recognized corrupted objects are not released. You can specify that error reports about unrecognized objects, corrupted objects, broken references, dangling references, and other release information is reported.

You use the @preview option to preview the results of the release operation prior to actually executing it. When you specify the @preview option, you view the preview report in the configuration monitor window.

The status of the release operation is monitored in real-time. Information about the release, whether it results from the normal monitoring process that occurs or from setting the @preview option is reported to the configuration monitor window. You can view the configuration monitor window by executing the interactive function \$show_monitor().

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

If you want to view the status reports of your toolkit operations in the transcript window, you can change the Pyxis Project Manager application defaults by executing the interactive function \$setup_config_monitor() from the popup command line or via its corresponding menu item **Setup > Monitor > Config_Monitor**. For complete information about specifying your session setup values, refer to “[Changing Your Session Setup](#)” in the *Pyxis Project Manager User's Manual*.

Examples

This example opens the *object_config* design object, then releases it to the *\$PROJ_DESIGN/design1/misc* target directory, and updates its references. The release is annotated with the “my_test_release” string. Both reference traversal and the preview switch are set to off. If the directory does not exist, this function creates it.

```
$open_configuration("$PROJ_DESIGN/design1/object_config", 0,  
    @read_write);  
  
$release_object(["$PROJ_DESIGN/design1/object_config",  
    "Mgc_file"], "$PROJ_DESIGN/design1/misc", "release_config",  
    "my_test_release", [], @nofollow, @nopreview, @create, @update);
```

Related Topics

[\\$\\$copy_configuration\(\)](#)

[\\$show_monitor\(\)](#)

[\\$setup_monitor\(\)](#)

\$\$remove_configuration_entry()

Scope: dm_config_tk

Removes the specified configuration entry from the active configuration.

Usage

`$$remove_configuration_entry("name", "type", version)`

Arguments

- **name**
A string that specifies the pathname of the configuration entry.
- **type**
A string that specifies the type of the configuration entry.
- ***version***
An integer that specifies the version number of the configuration entry. The default is 0, which represents the current version.

Return Values

VOID.

Description

If you omit the version argument, this function removes the current version of the entry.

You can remove both primary and secondary entries. Removing an entry from a configuration does not delete the design object from disk.

Examples

This example opens the configuration *config_new*, removes version 2 of the entry *base*, and then saves the configuration.

```
$$open_configuration("$PROJECT_XYZ/example/config_new",0,  
    @read_write);  
  
$$remove_configuration_entry("$PROJECT_XYZ/d1/base",  
    "Image_file", 2);  
  
$$save_configuration();
```

Related Topics

[\\$\\$add_configuration_entry\(\)](#)

\$remove_configuration_entry()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window and must select at least one configuration entry.

Removes the selected configuration entry from the active Configuration window.

Usage

\$remove_configuration_entry()

REMove COnfiguration Entry

Edit > Remove Entry

Configuration window popup menu > **Remove Entry**

Arguments

None.

Return Values

VOID.

Description

You can remove either primary or secondary entries. When you remove an entry from a configuration, that entry is not deleted from the disk, but removed from the configuration only.

 **Caution** The \$remove_configuration_entry() function does not ask for permission to remove the selected configuration entry.

Examples

1. This example removes the selected configuration entry from the active Configuration window.

```
$remove_configuration_entry();
```

2. This example removes the selected configuration entry by using command syntax from the popup command line.

```
rem co e
```

Related Topics

[\\$add_configuration_entry\(\)](#)

[\\$report_entry_verification\(\)](#)

[\\$report_entry_info\(\)](#)

\$remove_external_library()

Scope: dmgr_proj_tk

Removes an external library reference from a root hierarchical design object (HDO) such as a project or external library, and removes the external library from the root HDO's location map.

Usage

```
$remove_external_library("root_hdo_path", "ext_lib_path")
```

remexl

REMove EXternal Library

Edit > External/Logic Libraries

Arguments

- **root_hdo_path**
The path to the project or external library that should no longer include the external library at "ext_lib_path".
- **ext_lib_path**
The path to the external library to be removed.

Return Values

VOID.

Examples

The following example removes the external library located at */tmp/extlib* from the */tmp/project* project.

```
$remove_external_library("/tmp/project", "/tmp/extlib");
```

Related Topics

[\\$include_external_library\(\)](#)

\$remove_toolbox()

Scope: dmgr_toolbox_area

Prerequisite: To use this function, you must be in an active Tools window that currently shows toolboxes, and you must select at least one toolbox.

Removes the selected toolbox from the current toolbox search path.

Usage

\$remove_toolbox()

REMove TOolbox

Edit > Remove Toolbox

Toolboxes window popup menu > **Remove Toolbox**

Arguments

None.

Return Values

VOID.

Description

After you remove a toolbox from the current toolbox search path, the Pyxis Project Manager application no longer searches that toolbox when it determines which tools are currently available for use.

Caution



The \$remove_toolbox() function does not ask you to confirm the removal. Instead, it removes the selected toolbox.

Examples

1. This example removes the selected toolbox from the Tools window.

```
$remove_toolbox();
```

2. This example removes the selected toolbox by using command syntax from the popup command line.

```
rem to
```

Related Topics

[\\$add_toolbox\(\)](#)

[\\$set_toolbox_search_path\(\)](#)

[Toolbar Search Path Startup File](#)

[Toolboxes](#)

\$report_configuration_info()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window.

Displays the status of the active Configuration window.

Usage

\$report_configuration_info()

REPOrt COnfiguration Info

Report > Report Configuration Info

Configuration window popup menu > **Report Configuration Info** > **Report Configuration Info**

Arguments

None.

Return Values

A string that specifies the name of the window. For example, if no other information windows are open, this function returns “info”. If another information window is currently open, this function returns “info#2”.

Description

In the configuration's monitor window, the interactive function \$report_configuration_info() displays the status of the active configuration. This function displays the following types of information:

- **Configuration Object** — Absolute pathname of the configuration object.
- **Lock Status** — Whether the configuration is locked or *not* locked. Locking a configuration prevents others from accessing the design object versions that are part of the configuration.
- **Configuration Build Status** — The state of the configuration in respect to the last successful build operation. The three possible configuration build status' and their meanings are as follows:
 - a. “The last Build was completed successfully without errors.”

A successful build has been performed since the configuration was locked. No changes have been made to either the build rules or the configuration entries since the last successful build.
 - b. “A successful Build has not been done since being locked.”

A successful build has not been accomplished since the Configuration was locked.

- c. “The configuration Build is either out-of-date or incomplete.”

A successful build has been performed since the configuration was locked. Changes have been made to either the build rules or the configuration entries since the last successful build.

- **Freeze Status** — The configuration is frozen or *not* frozen. Freezing a configuration prevents the version depth mechanism from deleting those entries (design object versions) that are beyond the version depth of each design object that is part of the configuration.
- **Change Status** — The configuration has changed or has not changed since the last time you saved it. A change is defined as changing a build rule, adding or deleting an entry, or building the configuration.

This function reports the configuration's status in real-time to the configuration's monitor window. You can interrupt its execution at any time by pressing the Kill key. When you halt the report operation, a dialog box appears prompting you to abort or to continue the operation. If you choose to abort, the operation is halted.

You can press the ReportInfo key from the Configuration window, as a shortcut method for executing the `$report_configuration_info()` function, as shown in “[Key Names Mapped to Functions & Scopes](#)” on page 804.

Examples

1. This example displays the status of the active Configuration window, in the configuration's monitor window.

```
$report_configuration_info();
```

2. This example displays the status of the active configuration in the monitor window using command syntax from the popup command line.

```
rep co i
```

3. This example shows a sample configuration status report.

```
Report Info for Configuration: "$PROJ_XYZ/example/cp_config"  
The configuration is not Locked.  
The last Build was completed successfully without errors.  
The configuration is not Frozen.  
The configuration has not changed since last Saved.
```

Related Topics

[\\$report_configuration_references\(\)](#)

[\\$report_reference_info\(\)](#)

[\\$report_entry_info\(\)](#)

[\\$report_tool_info\(\)](#)

[\\$report_entry_verification\(\)](#)

[\\$report_version_info\(\)](#)

[\\$report_object_info\(\)](#)

\$\$report_configuration_references()

Scope: dm_config_tk

Reports the references of entries in the active configuration.

Usage

`$$report_configuration_references(“pattern”, which_refs)`

Arguments

- ***pattern***
An optional string that specifies the pattern used to filter the references that are reported. This function reports only references that match this pattern. Default: Empty string "".
- ***which_refs***
A switch that specifies whether to limit the report to broken references or to include all references. Choose one of the following:
 - **@all (-All)** — Reports all the references of the configuration entries. Default.
 - **@broken (-Broken)** — Reports only the broken references of the configuration entries.

Return Values

VOID.

Description

References are broken if the object being referenced does not exist or if the object's fileset is not complete. The references are reported in the session configuration monitor window.

Examples

This example reports all broken references in the active configuration that match the *\$PROJ_XYZ/proj/exp/saturn* pattern.

```
$$report_configuration_references("$PROJ_XYZ/proj/exp/saturn", @broken);
```

Related Topics

[\\$\\$remove_configuration_entry\(\)](#)

[\\$\\$report_entry_verification\(\)](#)

\$report_configuration_references()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window and must have at least one configuration entry.

Reports the references of entries in the active Configuration window.

Usage

\$report_configuration_references(*“pattern”*, *which_refs*)

REPort COnfiguration References *“pattern”* *which_refs*

Report > References

Configuration window popup menu > **Report Configuration Info** > **References**

Arguments

- *pattern*

A string that specifies the pattern used in order to filter the references that are reported. This function reports only references that match the pattern. The default is the empty string “”.

- *which_refs*

A switch that specifies whether to limit the report to broken references or to include all references. Choose one of the following:

- **@all (-All)** — Reports all the references of the configuration entries. Default.
- **@broken (-Broken)** — Reports only the broken references of the configuration entries.

Return Values

VOID.

Description

References are broken if the object being referenced does not exist or if the object’s fileset is not complete. You can verbose configuration monitoring on or off, prior to executing the \$report_configuration_references() function.

The following information is reported:

- **Entry** — The absolute pathname of the entry whose references are listed beneath it. The version number is enclosed in square brackets. The following additional information about each entry is displayed beneath the pathname. If a pattern argument is specified, only those references whose pathnames match the pattern are reported.
- **Entry Type** — The type of entry.
- **Reference's Pathname** — The absolute pathname of each reference of the entry. If the @broken switch is specified, only broken references are reported.

- **Reference's Object Type** — The type of the design object to which the reference points.
- **Total Number of References** — The total number of references found for the configuration, respective of the specified filters.
- **Total Number Of Objects** — The total number of entries in the configuration whose references have been reported.

This function reports the references of entries in real-time to the configuration's monitor window. You can interrupt its execution at any time by pressing the Kill key. When you interrupt the report operation, a dialog box appears prompting you whether to abort or to continue the operation. If you choose to abort, the operation is halted.

When the `$report_configuration_references()` function is invoked from a menu, a prompt bar is displayed, enabling you to specify the pattern you want to search for in order to filter the references that you want reported. However, when the function is invoked by using command syntax, a prompt bar is not displayed; in this case, any required arguments must be entered on the command line.

Examples

1. This example reports all references of the active configuration that match the `$PROJ_X/design/fs` pattern.

```
$report_configuration_references("$PROJ_X/design/fs", @all);
```

Reference information is reported as shown below:

```
Report Configuration References
Examining references on: "$PROJ_X/design/fs" of type "Mgc_file".
Found reference to: "$PROJ_X/ascii" of type "Mgc_container".
Found reference to: "$PROJ_X/frame_info" of type "Mgc_file".
Found 2 reference(s) on 1 object(s).
```

2. This example reports all broken references of the active configuration that match the "fs" pattern, by using command syntax from the popup command line.

```
rep co r $PROJ_X/design/fs -b
```

Related Topics

[\\$report_configuration_info\(\)](#)

[\\$report_entry_verification\(\)](#)

[\\$report_entry_info\(\)](#)

\$report_entry_info()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window and must have at least one entry selected.

Displays the status of the selected entries in the active Configuration window.

Usage

\$report_entry_info()

REPort ENtry Info

Report > Entry Info

Configuration window popup menu > **Report Configuration Info > Entry Info**

Arguments

None.

Return Values

VOID.

Description

If no entries are selected, an error message is displayed. This function displays the following information about each entry:

- The absolute pathname.
- The version number, enclosed in square brackets.
- The state of the entry, current or fixed.
- Whether the entry is a primary or secondary entry.
- Whether the entry is retargetable. An entry is retargetable if its parent directory is not part of the configuration.
- The entry's target path, which is the entry's destination during copy or release of the configuration. For non-retargetable entries, the target path is simply the entry's leaf name. For retargetable entries, the target path can be any string. For example, you can specify a different leaf name, or a different absolute or relative pathname for the entry. During the copy or release, the entry is copied to the path that you specified.
- The properties held by the selected entries.

This function reports the entry information in real-time to the configuration's monitor window. You can interrupt its execution at any time by pressing the Kill key. When you halt the report operation, a dialog box appears prompting you to abort or to continue the operation. If you choose to abort, the operation is halted.

Examples

1. This example displays the status of the selected entries from the active Configuration window, in the configuration's monitor window.

\$report_entry_info();

2. This example shows a sample portion of a configuration entry status report.

```
Report Entry Info for Configuration: "$PROJ_X/config4"

Entry path: $PROJ_X/design[1]
Entry type: Mgc_container
  Current version
  Primary entry
  Retargetable
    Target_path: "design"
  Properties:
    creating_tool: "Dme_config"
Entry path: $PROJ_Y/ascii/fs_summary[1]
Entry type: Mgc_file
  Current version
  Secondary entry
  Non-retargetable
  Properties:
    creating_tool: "Dme_config"
```

3. This example displays the status of the selected entries in the active configuration by using command syntax from the popup command line.

rep en i

Related Topics

\$report_configuration_info()	\$report_object_info()
\$report_configuration_references()	\$report_reference_info()
\$report_entry_info()	\$report_tool_info()
\$report_entry_verification()	\$report_version_info()

\$\$report_entry_verification()

Scope: dm_config_tk

Reports the integrity of the selected entries in the active configuration.

Usage

\$\$report_entry_verification()

Arguments

None.

Return Values

VOID.

Description

This function displays the following information:

- **Total Number of Entries** — A summary of the total number of entries in the configuration.
- **Number of Broken Entries** — The number of broken entries in the configuration. Broken entries are defined as entries whose target object or specified version is missing, as entries whose target object fileset is incomplete, or as entries whose objects cannot be connected.
- **Number of Out-of-date Entries** — The number of out-of-date entries. Out-of-date entries refer to those current entries whose stored version number is no longer the current version. The stored version number is the current version number of the entry at the time that the configuration is built. Rebuilding the configuration updates out-of-date entries.

The \$\$report_entry_verification() function reports entry verification in real-time to the configuration monitor window. You can interrupt the execution of this function at any time by pressing the Kill key. When you halt the report operation, a dialog box appears prompting you to abort or to continue the operation. If you choose to abort, the operation is halted.

Examples

This example displays the entry verification report for the active configuration.

```
$$report_entry_verification();
```

Related Topics

[\\$\\$report_configuration_references\(\)](#)

\$report_entry_verification()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window.

Reports the integrity of entries in the active configuration.

Usage

\$report_entry_verification()

REPort ENtry Verification

Report > Entry Verification

Configuration window popup menu > **Report Configuration Info > EntryVerification**

Arguments

None.

Return Values

VOID.

Description

This function displays the following information:

- **Total Number of Entries** — A summary of the total number of entries in the configuration.
- **Number of Broken Entries** — The number of broken entries in the configuration. Broken entries are defined as entries whose target object or specified version is missing, as entries whose target object fileset is incomplete, or as entries whose object cannot be connected.
- **Number of Out-of-date Entries** — The number of out-of-date entries. Out-of-date entries refer to those current entries whose stored version number is no longer the current version. The stored version number is the current version number of the entry at the time that the configuration is built. Rebuilding the configuration updates out-of-date entries.

The \$report_entry_verification() function reports entry verification in real-time, to the configuration's monitor window. You can interrupt the execution of this function at any time by pressing the Kill key. When you halt the report operation, a dialog box appears prompting you to abort or to continue the operation. If you choose to abort, the operation is halted.

Examples

1. This example displays the entry verification report for the active configuration, in the configuration's monitor window.

```
$report_entry_verification();
```

-
2. This example displays the entry verification report for the active configuration by using command syntax from the popup command line.

rep en v

-
-
3. This example shows a configuration entry verification report.

```
Report Entry Verification
Total entries: 39  Total broken: 0  Total out-of-date: 0
```

Related Topics

[\\$report_configuration_info\(\)](#)

[\\$report_configuration_references\(\)](#)

[\\$report_entry_info\(\)](#)

\$\$report_global_status()

Scope: dme_dn_tk

Reports any errors currently on the global toolkit status stack to the transcript window.

Usage

\$\$report_global_status()

Arguments

None.

Return Values

VOID.

Description

The toolkit function \$\$report_global_status() calls the Common User Interface error manager to report any errors that are currently on the global toolkit status stack. Error messages are sent to both the Pyxis Project Manager message area and the transcript window.

Examples

This example uses the \$\$report_global_status() function to report errors.

```
function $move_object(destination : string)
{
    $$clear_global_status();
    local object_vector = $get_selected_objects(@full);
    overwrite = @cancel;
    $$move_object(object_vector, destination, overwrite,
        @ncreate);
    if ($$get_status_code() != 0) {
        $$report_global_status();
        return;
    }
}
```

Related Topics

[\\$\\$clear_global_status\(\)](#)

[\\$\\$get_status_code_stack\(\)](#)

[\\$\\$get_status_code\(\)](#)

[\\$\\$get_status_messages\(\)](#)

\$report_object_info()

Scope: dmgr_model

Prerequisite: To use this function, you must be in an active Navigator window and must select at least one design object.

Displays information about the selected design object.

Usage

\$report_object_info()

REPort OBject Info

Report > Object Info

Arguments

None.

Return Values

A string that specifies the name of the window. For example, if no other information windows are open, this function returns “info”. If another information window is currently open, this function returns “info#2”.

Description

The interactive function \$report_object_info() displays, in a read-only window, information about the selected design object. To alert you to potential problems, in addition to the information listed below, this function reports missing fileset members.

You can press the ReportInfo key, as a shortcut method for executing the \$report_object_info() function, as shown in “[Key Names Mapped to Functions & Scopes](#)” on page 804.

This function displays the following information:

- **Object Name** — The name of the design object about which information is reported.
- **Object Type** — The type of the design object.
- **Location** — The absolute pathname of the design object.
- **Protection** — The protection status of the design object. In this string, “r” means read, “w” means write, “x” means execute, and “-” means no permission. From left to right, the first three characters denote your permissions, the second three denote your group's permissions, and the last three denote all other's permissions.
- **Lock Status** — The design object as either locked or unlocked. When a design object is locked, it cannot be opened for editing. The locked status can indicate a write lock, in which case you cannot access the design object, or a read lock, in which case you can read, but not edit, the design object.

- **Released Status** — The design object as either released or unreleased. If the design object is a released, versioned object, it cannot be opened for editing. This read-only state is independent of read and write locks, and unlike locks, its state is permanent.
- **Last Revision** — The date that the object was last edited.
- **Current Version Number** — The number of the design object's current version. This field appears only for versioned design objects.
- **Version Depth** — The version depth of the design object. This field appears only for versioned design objects. This item indicates how many versions a design object keeps of itself before discarding the oldest version. For versioned design objects, a value of -1 signifies that there is an infinite version depth; versions are never pruned off.
- **Object Properties** — Properties of the design object, if any exist. These properties are attached to the entire design object and accumulate as the object evolves. The properties are shown in the following format:

 name: "value"
- **Fileset Members** — Fileset members of the design object. If a required fileset member for the selected object is missing, it is reported.

Examples

1. This example displays information about the selected design object.

\$report_object_info();

2. This example is a design object information report.

```
Object Name:  add_det
Object Type:  mgc_component
Location:     $PROJ_Y/training/qsim_n/add_det
Protection:   rwxrwxrwx
Lock Status:  Unlocked
Released Status:  Unreleased
Date Last Modified:  Fri Jul 31 09:58:51 2012
Current Version Number:  1
Version Depth:  2
Object Properties:
    This object has no properties.
Fileset Members:
    add_det  (directory)
```

3. This example displays information about the selected design object by using command syntax from the popup command line.

rep ob i

Related Topics

[\\$report_reference_info\(\)](#)

[\\$report_version_info\(\)](#)

[\\$report_tool_info\(\)](#)

[Versions](#)

[Filesets](#)

[Types](#)

[Changing Version Depth](#)

\$report_reference_info()

Scope: dmgr_reference_model

Prerequisite: To use this function, you must be in an active Reference window and must select at least one reference.

Displays information about the selected reference.

Usage

\$report_reference_info

REPort REference Info

Report > Reference Info

References window popup menu > **Report Reference Info**

Arguments

None.

Return Values

A string that specifies the name of the window. For example, if no other information windows are open, this function returns “info”. If another information is currently open, this function returns “info#2”.

Description

The interactive function \$report_reference_info() displays, in a read-only window, information about the selected reference. This function displays the following information:

- **Object Name** — The name of the design object to which the reference belongs.
- **Object Type** — The type of the design object to which the reference belongs.
- **Location** — The absolute pathname of the design object to which the reference belongs.
- **Reference Path** — The absolute pathname of the referenced design object.
- **Reference Type** — The type of the referenced design object.
- **Referenced Version Number** — The version number of the referenced design object to which this reference points. If this reference points to a *fixed* version, then this number might not be the current version number of the referenced design object.
- **Reference path type** — The reference path type of the referenced design object, either soft or hard:
 - A hard reference path is one that begins with a "/".
 - A soft reference path is one that begins with a "\$".
- **Reference state** — The state of the referenced design object. Every reference can have one of the following states:

- **Current.** The reference points to the current version of the referenced object. As that object evolves, the reference still always points to the most current version.
- **Fixed.** The reference points to a particular, previous version of the referenced object. As that object evolves, this reference still points to the same version.
- **Read-only.** An application that accesses the referenced object through a read-only reference can read, but not edit, that object.
- **Reference Properties** — The properties held by the reference, if any exist. The properties are shown in the following format:

name: “value”

You can press the ReportInfo key from the Reference window, as a shortcut method for executing the \$report_reference_info() function, as shown in “[Key Names Mapped to Functions & Scopes](#)” on page 804.

Examples

1. This example displays information about the selected reference.

```
$report_reference_info();
```

2. This example displays information about the selected reference by using command syntax from the popup command line.

```
rep re i
```

3. This example is a portion of a sample reference information report.

```
Object Name:  config4
Object Type:  Dme_config_do
Location:     $PROJ_X/config4

Reference Path:  $PROJ_X/design
Reference Type:  Mgc_container
Referenced Version Number:  1
Reference path type:  Soft
Reference state:  Access the current version.
Reference Properties:
    creating_tool:  "Dme_config"

Reference Path:  $PROJ_X/frame_info
Reference Type:  Mgc_file
Referenced Version Number:  1
Reference path type:  Soft
Reference state:  Access to a specified (fixed) version.
Reference Properties:
    creating_tool:  "Dme_config"
```

Related Topics

[`\$add_reference\(\)`](#)

[`\$delete_reference\(\)`](#)

[`\$show_references\(\)`](#)

[References](#)

\$report_tool_info()

Scope: dmgr_tool_area

Prerequisite: To use this function, you must be in the Tools window and must select at least one tool.

Displays information about the selected tool.

Usage

\$report_tool_info()

REPort TOol Info

Report > Tool Info

Tools window popup menu > **Report Tool Info**

Arguments

None.

Return Values

A string that specifies the name of the window. For example, if no other information windows are open, this function returns “info”. If another information window is currently open, this function returns “info#2”.

Description

The interactive function \$report_tool_info() displays, in a read-only window, information about the selected tool.

You can press the ReportInfo key from the Tools window, as a shortcut method for executing the \$report_tool_info() function, as shown in “[Key Names Mapped to Functions & Scopes](#)” on page 804.

Examples

1. This example displays information about the selected tool.

```
$report_tool_info();
```

2. This example displays information about the selected tool by using command syntax from the popup command line.

```
rep to i
```

3. This example is a tool information report.


```

Tool Name: Editor
Tool Type: Mgc_default_ascii_editor
Location: $MGC_HOME/toolbox/Editor
Protection: rwxrwxrwx
Date Last Modified: Fri Sep 18 10:23:30 2012
Current Version Number: 1
Version Depth: 2
Object Properties:
    This object has no properties.
Type Properties:
    default_tool: "Mgc_default_ascii_editor"
    large_icon: ["$MGC_HOME/registry/fonts/dm_object.icons",
"1"]
    small_icon: ["$MGC_HOME/registry/fonts/dm_text.icons", "1"]
    ddms_versioned_object: ""
    input_data: ["Mgc_file"]
    tool_version: "V1.00"
    creation_data: "10-SEP-12"
    attribute_file_required: "TRUE"
    fileset_def: ["!r!c"]

```

Related Topics

[\\$report_configuration_info\(\)](#)

[\\$report_reference_info\(\)](#)

[\\$report_object_info\(\)](#)

[\\$report_version_info\(\)](#)

\$report_type_info()

Scope: dmgr_type_area

Prerequisite: To use this function, you must be in an active type window and must select at least one type.

Displays the properties of the selected type.

Usage

\$report_type_info()

REPort TYpe Info

Report > Type Info

Types window popup menu > **Report Type Info**

Arguments

None.

Return Values

A string that specifies the name of the read-only window that is displayed. For example, if no other read-only windows are open, this function returns “info”. If another read-only window is open, this function returns “info#2”.

Description

The interactive function \$report_type_info() displays a Type Information Report in a read-only window. This report contains a list of properties for the selected type. The list contains the property name, the AMPLE type, and the value. To edit the properties of a type, you must use the Pyxis Registrar application. You cannot edit type properties from the Pyxis Project Manager application.

Examples

1. This example displays the properties of the selected type.

```
$report_type_info();
```

2. This example displays the selected type's properties by using command syntax from the popup command line.

```
rep ty i
```

Related Topics

[\\$load_registry\(\)](#)

[\\$open_types_window\(\)](#)

[Types](#)

\$report_version_info()

Scope: dmgr_version_model

Prerequisite: To use this function, you must be in a version window and must select exactly one version.

Displays information about the selected version.

Usage

\$report_version_info()

REPOrt VErsion Info

Report > Version Info

Versions window popup menu > **Report Version Info**

Arguments

None.

Return Values

A string that specifies the name of the window. For example, if no other information windows are open, this function returns “info”. If another information window is currently open, this function returns “info#2”.

Description

The interactive function \$report_version_info() displays, in a read-only window, information about the specified version. This function displays the following information:

- **Object Name** — The name of the design object of which this is a single version.
- **Object Type** — The type of the design object of which this is a single version.
- **Location** — The absolute pathname of the design object of which this is a single version.
- **Object Version** — The version number. This is an integer that is attached to each saved state (version) as the design object evolves. The first version of an object is always version number 1, the second is number 2, and so on.
- **Version Frozen?** — The state of the version. A frozen version is not pruned as the design object evolves.
- **Size** — The size of the version.
- **Version Object Properties** — The properties of the design object of which this is a single version, if any exist. The properties are shown in the following format:

name: “value”

These properties are attached to the entire design object and accumulate as the object evolves.

- **Version Properties** — The properties of the version, if any exist. The properties are shown in the following format:

name: “value”

These properties are attached to this particular version. Unlike object properties, you can view version properties only in this information window.

- **Version References** — The references of the version, if any exist. The references are shown as the pathnames of the referenced objects. These references reflect the references that the design object held when this version was created. The current version of the design object may have quite a different set of references. When you select the whole design object in the Navigator window and show its references, you show the references held by the current version of the object.

You can press the ReportInfo key from the version window, as a shortcut method for executing the \$report_version_info() function, as shown in “[Key Names Mapped to Functions & Scopes](#)” on page 804.

Examples

1. This example displays information about the selected version.

```
$report_version_info();
```

2. This example displays information about the selected version by using command syntax from the popup command line.

```
rep ve i
```

3. This example shows a sample version report.

```
Object Name:  config3
Object Type:  Dme_config_do
Location:     $PROJ_X/config3
Object Version:  1
Version Frozen?  No
Size:  unavailable
Version Object Properties
  This version has no properties,
Version Properties:
  This version has no properties,
Version References:
  $PROJ_Y/conn:  Mgc_file
  $PROJ_X/config3:  Dme_config_do
```

Related Topics

[\\$report_configuration_info\(\)](#)

[\\$report_reference_info\(\)](#)

[\\$report_object_info\(\)](#)

[\\$report_tool_info\(\)](#)

[Versions](#)

\$\$resolve_path()

Scope: dme_dn_tk

Resolves a specified pathname into a hard pathname that does not have a symbolic link in its path.

Usage

```
$$resolve_path("path_name")
```

Arguments

- **path_name**

A string that specifies the pathname to be resolved. You may enter a hard, soft, or relative pathname.

Return Values

The equivalent hard pathname that does not have a symbolic link in its path. If an error is encountered, an empty path is returned.

Description

If you specify a soft or a relative pathname, this function converts the pathname to an absolute pathname which does not contain a soft prefix or special characters, prior to removing its symbolic link.

If you specify a pathname that does not begin with either a dollar sign (\$) or a slash (/), this function interprets the pathname as a relative pathname and first generates an absolute pathname by removing any "." and ".." and special character sequences. Pathname arguments of the form "anything/" are converted to the form "anything". Pathname arguments of the form "/." are converted to the form "/". Pathname arguments of the form "/anything/.." are converted to the form "/".

If you specify a pathname that begins with a dollar sign (\$), this function first replaces the soft pathname with its equivalent hard pathname, and then removes any "." and ".." sequences.

This function then verifies that the absolute pathname you specified, or the pathname that resulted from the conversion of a soft or relative pathname, is a hard pathname. This function then returns an equivalent hard pathname which does not have a symbolic link in its path.

Examples

This example shows the resolution of the *\$PROJ_X/design/dir1/link* absolute pathname, where the "link" leaf is a link that points to the */usr/bob/xxx* absolute hard pathname.

```
$$resolve_path("$PROJ_X/design/dir1/link");
```

This function call returns the */usr/bob/xxx* resolved absolute pathname.

Related Topics

[\\$\\$fix_relative_path\(\)](#)

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

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

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

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

[\\$\\$is_relative_path\(\)](#)

[\\$get_navigator_directory_hard\(\)](#)

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

\$revert_object_from_rc()

Scope: dmgr_project_model

Reverts the object specified by the path and type to the specified revision and timestamp.

Usage

`$revert_object_from_rc("path", "type", "revision", "timestamp", read_only, recurse)`

Arguments

- **path**
A string that specifies the object path to revert.
- **type**
A string that specifies the object type to revert.
- **revision**
A string that specifies the revision to use when reverting the object.
- **timestamp**
A string that specifies the timestamp to use when reverting the object.
- **read_only**
A Boolean that specifies whether the object should be read-only after revert finishes. Default is @true.
- **recurse**
A Boolean that specifies whether revert occurs hierarchically. Default: @true.

Return Values

VOID

Description

The `read_only` argument specifies if the object should be read-only after revert finishes.

Examples

```
$revert_object_from_rc("$PROJECT/lib/cell/symbol", "mgc_symbol", "8", "2009-01-07T00:41:56.339815z", @true);
```

Related Topics

[\\$add_project_to_rc\(\)](#)

[\\$create_work_area_from_rc\(\)](#)

[\\$cancel_checkout_objects_for_rc\(\)](#)

[\\$checkin_objects_to_rc\(\)](#)

[\\$checkout_objects_from_rc\(\)](#)

[\\$update_objects_from_rc\(\)](#)

`$object_status_for_rc()`

\$\$revert_version()

Scope: dme_do_tk

Deletes the current version of the specified design object and makes the most recent version the new current version.

Usage

```
$$revert_version("obj_name", "obj_type")
```

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **obj_type**
A string that specifies the type of the design object.

Return Values

VOID.

Description

The specified design object must be a versioned design object. After the deletion, the most recent version available becomes the current version. For safety, this function cannot delete a version of a design object that either has only one version or the current version is frozen.

Examples

This example deletes the current version of the “canvas” design object and makes the most recent version available as the new current version.

```
$$revert_version("$PROJECT_XYZ/project/canvas", "Mgc_file");
```

Related Topics

[\\$\\$delete_version\(\)](#)

[\\$\\$get_version_properties\(\)](#)

[\\$\\$freeze_version\(\)](#)

[\\$\\$set_version_depth\(\)](#)

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

[\\$\\$unfreeze_version\(\)](#)

[\\$\\$get_version_depth\(\)](#)

\$revert_version()

Scope: dmgr_model

Prerequisite: To use this function, you must be in an active Navigator window and must select at least one versioned design object.

Deletes the current version of the selected design object and makes the most recent version the new current version.

Usage

\$revert_version()

REVert VErSION *force*

Edit > Revert Version

Navigator window popup menu > **Revert Version**

Arguments

- *force (-Force)*

A switch name that enables you to suppress the dialog box. This switch is available only when you execute this function from the command line, by using command syntax.

Return Values

VOID.

Description

The selected design object must be a versioned design object. After this function executes, the most recent version available becomes the current version. This function cannot delete a version of a design object that has only one version.

If you select the **Revert Version** menu item, a dialog box appears asking for permission to continue. To suppress the dialog box, you execute this function from the command line by using command syntax, and then specifying the @force switch.

Examples

1. This example reverts the selected version of the design object.

```
$revert_version();
```

2. This example reverts a version and suppresses the dialog box by using command syntax from the popup command line.

```
rev ve -force
```

Related Topics

[\\$\\$delete_version\(\)](#)

[Versions](#)

Chapter 5

Function Dictionary Part 4

Temporary short description for converting multi .fm chapter.

\$\$salvage_object()

Scope: dme_do_tk

Attempts to repair a design object that has existing session data or malformed locks resulting from an application failure.

Usage

```
$$salvage_object("obj_name", "type", save_pending_data)
```

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **obj_type**
A string that specifies the type of the design object.
- **save_pending_data**
A Boolean that specifies whether this function uses or discards session data when repairing the design object. Choose one of the following:
 - **@true** — Uses the session data when attempting to repair the design object. Session data files have the file name extension *.ed* and are the result of doing a write operation. These data files are used in order to form a new version of the object. If you exit your application before saving, these files exist but are not saved as part of the design object.
 - **@false** — Discards the session data when attempting to repair the design object. The design object is restored to its original state at the time of the last save operation.

Return Values

VOID.

Description

During the repair operation, this function removes any malformed lock files that are found. The “save_pending_data” argument determines whether the repair operation uses session data when attempting to repair the damaged object.

Examples

This example attempts to repair the *my_file* design object, discarding its session data.

```
$$salvage_object("$PROJECT_XYZ/design1/my_file", "Mgc_file", @false);
```

Related Topics

[\\$\\$delete_object\(\)](#)

\$salvage_object()

Scope: dmgr_model

Prerequisite: To use this function, you must be in an active Navigator window and must select at least one design object.

Attempts to repair a design object that has existing session data or malformed locks resulting from an application failure.

Usage

`$salvage_object(save_edits)`

SALvage OBJect save_edits

File > Salvage Object

Arguments

- **save_edits**

A switch name that specifies whether this function uses or discards session data when repairing the design object.

- **@save (-Save)** — Uses the session data when attempting to repair the design object. Session data files have the filename extension *.ed*, and result from a write operation. If you exit your application saving, these files exist but are not saved as part of the design object. Default.
- **@nosave (-Nosave)** — Discards the session data when you attempt to repair the design object. The design object is restored to its original state at the time of the last save operation.
- **@ask_save (-Ask_save)** — Brings up a dialog box that asks whether to save or to discard session data.

Return Values

VOID.

Description

During the repair operation, this function removes any malformed lock files that are found. The `save_edits` argument determines if the repair operation uses session data when attempting to repair the damaged object.

Examples

1. This example attempts to repair the selected design object, discarding session data.

`$salvage_object(@nosave);`

2. This example attempts to repair the selected design object by using command syntax from the popup command line.

sal ob -nosave

\$\$save_configuration()

Scope: dm_config_tk

Writes the active configuration to the disk and increments the version number of the configuration object.

Usage

\$\$save_configuration()

Arguments

None.

Return Values

VOID.

Description

To save modifications of an existing configuration under a new name you use the \$\$save_configuration_as() function.

Note



If you want to save your edits, you must execute this function prior to closing the configuration. If you execute the \$\$close_configuration function prior to executing either the \$\$save_configuration() or the \$\$save_configuration_as() function, changes to the configuration are not saved.

Examples

This example opens the “my_config” configuration, adds the “base” entry, and then saves the configuration.

```
$$open_configuration("$PROJECT_XYZ/design1/my_config", 0, @read_write);  
$$add_configuration_entry("$PROJECT_XYZ/design2/base", "Mgc_file", 2);  
$$save_configuration();
```

Related Topics

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

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

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

[\\$\\$save_configuration_as\(\)](#)

\$save_configuration()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window.

Writes the entries in the active Configuration window to disk and increments the version number of the configuration object.

Usage

\$save_configuration()

SAVe COnfiguration

File > Save

Arguments

None.

Return Values

VOID.

Description

To save modifications of an existing configuration under a new name, you use the \$save_configuration_as() function.

Examples

1. This example saves the contents of the active Configuration window.

```
$save_configuration();
```

2. This example saves the contents of the active Configuration window by using command syntax from the popup command line.

```
sav co
```

Related Topics

[\\$open_configuration_window\(\)](#)

[\\$save_configuration_as\(\)](#)

\$\$save_configuration_as()

Scope: dm_config_tk

Saves the active configuration under a new name.

Usage

```
$$save_configuration_as("path")
```

Arguments

- **path**

A string that specifies the pathname where the configuration is saved. You can specify a relative or an absolute pathname. If you specify a relative pathname, the Pyxis Project Manager application assumes the path is relative to the current working directory.

Return Values

VOID.

Description

The toolkit function `$$save_configuration_as()` saves the active configuration at a location specified by the “path” argument. After this function executes, the configuration object itself is added to the configuration.

Note

If you want to save your edits, you must execute this function prior to closing the configuration. If you run the `$$close_configuration()` function prior to running either the `$$save_configuration()` or the `$$save_configuration_as()` function, changes to the configuration are not saved.

Examples

This example opens the “config_1” configuration, removes the “plan” entry, and then saves it as “config_2”.

```
$$open_configuration("/user/mary/project/config_1", 0, @read_write);
```

```
$$remove_configuration_entry("/user/mary/project/plan");
```

```
$$save_configuration_as("/user/mary/project/config_2");
```

Related Topics

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

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

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

[\\$\\$save_configuration\(\)](#)

\$save_configuration_as()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window.

Saves the entries in the active configuration under a new configuration object name.

Usage

`$save_configuration_as("pathname")`

SAVe COnfiguration As pathname

File > Save As

Configuration window popup menu > **Save As**

Arguments

- **pathname**

A string that specifies the pathname where the configuration is saved. You must specify an absolute pathname.

Return Values

VOID.

Description

The interactive function `$save_configuration_as()` saves the contents of the active Configuration window at a location specified by the `pathname` argument. If you do not specify the absolute pathname, this function saves the configuration in the default working directory. After saving the configuration, this function adds the configuration object to the active Configuration window. The configuration object itself becomes part of the configuration.

Examples

1. This example saves the contents of the active Configuration window.

```
$save_configuration_as("$PROJ_XYZ/proj/my_config");
```

2. This example saves the contents of the active Configuration window by using command syntax on the popup command line.

```
sav co a $PROJ_XYZ/proj/my_config
```

Related Topics

[\\$open_configuration_window\(\)](#)

[\\$save_configuration\(\)](#)

\$\$save_object()

Scope: dme_do_tk

Saves the specified locked design object.

Usage

```
$$save_object("obj_name", "type")
```

Arguments

- **obj_name**
A string that specifies the pathname of the locked design object.
- **obj_type**
A string that specifies the type of the locked design object.

Return Values

VOID.

Description

The toolkit function `$$save_object()` saves the specified design object previously locked by the `$$lock_object()` function. The functions `$$save_object()`, `$$lock_object()`, and `$$unlock_object()` can be used when you perform multiple operations on a single design object. You can lock the design object once, perform all your operations (saving as needed), and then unlock object when you are finished.

This function does not work with the following tool viewpoint functions `$$create_versioned_object()`, `$$open_versioned_object()`, and `$$close_versioned_object()`.

The `$$save_object()` function performs a specific action when you save the design object, depending on which lock mode you chose for the `$$lock_object()` function.

1. If you specified the lock as `@read_only`, this function does nothing.
2. If you specified the lock as `@annotate`, this function only overwrites the attribute (`.attr`) file.
3. If you specified the lock as `@create_version`, this function overwrites the attribute (`.attr`) file and increments the version number of the object. If `.ed` working copies exist, they are incorporated into the new version's data members.

Examples

This example saves the "schedule" design object.

```
$$save_object("$PROJ_XYZ/proj/schedule", "Mgc_file");
```

Related Topics

[\\$\\$lock_object\(\)](#)

[\\$\\$unlock_object\(\)](#)

\$save_toolbox_search_path()

Scope: dmgr_tool_window

Prerequisite: To use this function as a command or function, you must be in an active Tools window that is in either tools or toolbox mode. To use this function through a menu selection, you must be in an active Tools window that is in toolbox mode.

Saves the toolbox search path.

Usage

\$save_toolbox_search_path()

SAVe TOolbox Search Path

Edit > Save

Arguments

None.

Return Values

VOID.

Description

The interactive function \$save_toolbox_search_path() saves the toolbox search path in the *dmgr_toolbox_path.startup* file, which is located under your home directory at the */mgc/startup* location. The toolbox search path consists of the list of pathnames displayed in the toolbox window.

These saved pathnames are exactly as you specified them when you added them. The only exception is in the case of relative pathnames. If you entered a relative pathname for a toolbox, that toolbox's pathname is considered relative to the working directory. However, this pathname is still consistent with its pathname as it appears in the toolbox window.

In subsequent invocations of the Pyxis Project Manager application, the *dmgr_toolbox_path.startup* file recreates the toolbox search path that is displayed in the toolbox window. If the Pyxis Project Manager application is invoked with the *-nostart* switch, none of the startup files are executed. Therefore, any changes made to the toolbox search path are not effective.

If you change the toolbox search path and do not save it, when you exit the Tools window, a dialog box appears with the following prompt:

"Toolbox Search Path has Changed. Do you want to save?"

If you choose "Yes", this function executes and the new toolbox search path is saved. If you choose "No", the new toolbox search path is discarded.

Examples

1. This example explicitly saves the toolbox search path.

```
$save_toolbox_search_path();
```

2. This example explicitly saves the toolbox search path, by using command syntax from the popup command line.

```
sav to s p
```

Related Topics

[\\$add_toolbox\(\)](#)

[\\$set_toolbox_search_path\(\)](#)

[\\$get_toolbox_search_path\(\)](#)

[\\$view_toolboxes\(\)](#)

[\\$remove_toolbox\(\)](#)

[\\$view_tools\(\)](#)

\$search()

Scope: dm_iconic_area or dm_list_area

Prerequisite: To use this function, you must be in an active Navigator window, Configuration, Tools, or Trash window.

Searches the active window for the specified pattern.

Usage

`$search("pattern")`

SEArch pattern

Navigator window > **File** > **Explore** > **Search**

Session window > **File** > **Explore** > **Search**

Tools window **Edit** > **Search**

Trash window > **Edit** > **Search**

Configuration window popup menu > **Search**

Arguments

- **pattern**

A string that specifies the pattern for which this function searches. You can specify the string using UNIX System V regular expressions.

Return Values

VOID.

Description

The interactive function `$search()` searches for and highlights, in red line, the first design object whose name matches the pattern specified in the pattern argument. You may specify the pattern by using UNIX System V regular expressions and case sensitivity. The design object you search for must be within the currently active directory.

Examples

1. This example searches for the "schedule" design object in the current window.

```
$search("schedule");
```

2. This example searches for the same design object, by using command syntax from the popup command line.

```
sea schedule
```

3. This example searches for all design objects with a *.lib* extension, by using a regular expression.

sea ^.*\.lib\$

Related Topics

[\\$search_again\(\)](#)

\$search_again()

Scope: dm_iconic_area

Prerequisite: To use this function, you must be in an active iconic Navigator, Trash, or Tools window.

Repeats the search of the most recent search pattern specified.

Usage

`$search_again()`

SEArch AGain

Arguments

None.

Return Values

VOID.

Examples

1. This example repeats the previous search.

`$search_again();`

2. This example repeats the previous search by using command syntax from the popup command line.

`sea ag`

Related Topics

[\\$search\(\)](#)

\$select_all()

Scope: dm_config_window, dm_iconic_area, or dm_list_area

Prerequisite: To use this function, you must be in an active Navigator, Reference, Tools, Trash, Version, or Configuration window.

Selects all design objects in the active window.

Usage

\$select_all()

SElect All

Navigator window > **File** > **Select** > **All**

Configuration window > **File** > **Select** > **All**

Arguments

None.

Return Values

VOID.

Description

To unselect all design objects, use the \$unselect_all() function.

You can press the SelectAll key, as a shortcut method for executing the \$select_all() function, as shown in “[Key Names Mapped to Functions & Scopes](#)” on page 804.

Examples

1. This example selects all design objects in the active window.

```
$select_all();
```

2. This example selects all design objects in the active window by using command syntax from the popup command line.

```
sel al
```

Related Topics

[\\$select_config_entry\(\)](#)

[\\$select_reference\(\)](#)

[\\$select_tool\(\)](#)

[\\$select_trash_object\(\)](#)

[\\$select_version\(\)](#)

[\\$unselect_all\(\)](#)

[\\$unselect_trash_object\(\)](#)

\$select_by_name()

Scope: dmgr_iconic_area, dmgr_list_area, or dm_config_window

Prerequisite: To use this function, you must be in either an active Navigator or an active Configuration window.

Selects a design object based on its name.

Usage

Navigator window:

```
$select_by_name("name", "type", mode)
SElECT BY Name name type mode
```

Configuration window:

```
$select_by_name("name", "type", version, mode)
SElECT BY Name name type version mode
```

Object > Select > By Name

Navigator window > **File** > **Select** > **By Name**

Configuration window > **File** > **Select** > **By Name**

Arguments

- **name**
A string that specifies the absolute pathname or the leaf name of the design object. You can specify the name by using UNIX System V wildcards.
- **type**
A string that specifies the type of the design object. You can specify the type by using UNIX System V wildcards.
- **version**
An integer that specifies the version number of the configuration entry.

Note



This argument is only valid in the Configuration window.

If the version is not specified, by default, the version is ignored and all entries that match the specified name and type are selected.

- **mode**
A switch name that indicates whether other design objects remain selected. Choose one of the following:
 - **@reselect (-Reselect)** — Unselects all currently selected design objects and selects the specified design object. Default.
 - **@add (-Add)** — Adds the specified design object to the current selection set.

Return Values

VOID.

Description

If you omit either the type or mode argument, this function selects all design objects that match the pattern of the argument specified.

The version argument is only valid in the Configuration window. If the version argument is not specified, by default, the version is ignored and all entries that match the name and type arguments are selected. To select a specific version, you must specify the version argument.

To add the selected design object to the current selection set, you use the @add value; otherwise, this function selects the design object and unselects any previous selections.

Examples

1. This example selects the “logo” design object and adds it to the current selection set.

```
$select_by_name("logo", "Mgc_file", @add);
```

2. This example selects version 3 of the “logo” design object and adds it to the current selection set by using command syntax from the popup command line.

```
sel by n logo Mgc_file 3 -add
```

3. This example uses a wildcard to select all design objects that begin with “config”.

```
sel by n config*
```

Related Topics

[\\$select_all\(\)](#)

[\\$select_by_type\(\)](#)

[\\$select_config_entry\(\)](#)

[\\$select_reference\(\)](#)

[\\$select_version\(\)](#)

[\\$select_tool\(\)](#)

[\\$select_trash_object\(\)](#)

[\\$unselect_all\(\)](#)

[\\$unselect_by_name\(\)](#)

[\\$unselect_by_type\(\)](#)

[\\$unselect_trash_object\(\)](#)

\$select_by_library()

Prerequisite: To use this function, you must be in an active Configuration window.

Enables you to select from the configuration entries by library type.

Usage

`$select_by_library()`

File > Select > By Library

Arguments

None.

Return Values

VOID.

Description

This function is dependent on having the location map entry -t(type) for libraries active in the your session location map. When activated, a list box appears from which you can make the selections, and once accepted, the selections are applied to the configuration.

Examples

`$select_by_library()`

Related Topics

[\\$select_all\(\)](#)

\$select_by_type()

Scope: dmgr_iconic_area, dmgr_list_area, or dm_config_window

Prerequisite: To use this function, you must be in either an active Navigator or an active Configuration window.

Selects the specified design object based on its type.

Usage

`$select_by_type("type", mode)`

SElect BY Type *type mode*

Navigator window > **File** > **Select** > **By Type**

Configuration window > **File** > **Select** > **By Type**

Arguments

- *type*

A string that specifies the type of the design object. You can specify the type by using UNIX System V wildcards.

- *mode*

A switch name that indicates whether other design objects remain selected. Choose one of the following:

- **@reselect (-Reselect)** — Unselects all currently selected design objects and selects the specified design objects. Default.
- **@add (-Add)** — Adds the specified design object to the current selection set.

Return Values

VOID.

Description

To add the selected design object to the current selection set, you use the @add value; otherwise, this function selects the design object and unselects any previous selections.

Examples

1. This example selects all design objects of the "Mgc_file" type and adds them to the current selection set.

```
$select_by_type("Mgc_file", @add);
```

2. This example selects the same design object by using command syntax from the popup command line.

```
sel by t Mgc_file -add
```


3. This example uses a wildcard to select all design objects that have a type name that begins with the “Mgc” string.

sel by t Mgc*

Related Topics

[\\$select_all\(\)](#)

[\\$select_by_name\(\)](#)

[\\$select_config_entry\(\)](#)

[\\$select_reference\(\)](#)

[\\$select_tool\(\)](#)

[\\$select_trash_object\(\)](#)

[\\$select_version\(\)](#)

[\\$unselect_all\(\)](#)

[\\$unselect_by_name\(\)](#)

[\\$unselect_by_type\(\)](#)

[\\$unselect_trash_object\(\)](#)

\$select_config_entry()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window.

Selects the specified configuration entry.

Usage

`$select_config_entry("name", "type", version, mode)`

SElect COnfig Entry *name type version mode*

Arguments

- **name**

A string that specifies the name of the configuration entry. You must specify the absolute pathname of the configuration entry. You can specify the name by using UNIX System V wildcards.

- **type**

A string that specifies the type of the configuration entry. You can specify the type by using UNIX System V wildcards.

- ***version***

An integer that specifies the version number of the configuration entry. If the version is not specified, by default, the version is ignored and all entries that match the specified name and type are selected.

- ***mode***

A switch name that indicates whether other design objects remain selected. Choose one of the following:

- **@reselect (-Reselect)** — Unselects all currently selected configuration entries and selects the specified entry. Default.
- **@add (-Add)** — Adds the specified configuration entry to the current selection set.

Return Values

VOID.

Description

You must specify the absolute pathname of the configuration entry.

If the version argument is not specified, by default, the version is ignored and all entries that match the name and type argument are selected. To select a specific version, you must specify the version argument.

To add the selected configuration entry to the current selection set, you use the @add value. Otherwise, this function selects the specified configuration entry and unselects any previous selections.

You can press the Select key, as a shortcut method for executing the \$select_config_entry() function, as shown in “[Key Names Mapped to Functions & Scopes](#)” on page 804.

Examples

1. This example selects all versions of the “add_det” configuration entry and unselects any previously selected design objects.

```
$select_config_entry("$PRJ_X/add_det", "mgc_components");
```

2. This example selects version 4 of the “solar_system” configuration entry and adds it to the current selection set.

```
$select_config_entry("$PROJECT_XYZ/d1/solar_system",  
"Image_file", 4, @add);
```

3. This example selects the same configuration entry, by using command syntax from the popup command line.

```
sel co e $PROJECT_XYZ/d1/solar_system Image_file 0 -reselect
```

Related Topics

[\\$select_all\(\)](#)

[\\$select_by_name\(\)](#)

[\\$select_by_type\(\)](#)

[\\$select_reference\(\)](#)

[\\$select_version\(\)](#)

[\\$select_tool\(\)](#)

[\\$unselect_all\(\)](#)

[\\$unselect_by_name\(\)](#)

[\\$unselect_by_type\(\)](#)

[\\$unselect_config_entry\(\)](#)

[\\$unselect_trash_object\(\)](#)

[Managing Designs](#)

\$select_object()

Scope: dmgr_iconic_area or dmgr_list_area

Prerequisite: To use this function, you must be in an active Navigator window.

Selects the specified design object.

Usage

`$select_object("name", "type", mode)`

SElect OBject name *type* *mode*

Arguments

- **name**

A string that specifies the absolute pathname or the leaf name of the design object. You can specify the name by using UNIX System V wildcards.

- **type**

A string that specifies the type of the design object. You can specify the type by using UNIX System V wildcards.

- **mode**

A switch name that indicates whether other design objects remain selected. Choose one of the following:

- **@reselect (-Reselect)** — Unselects all currently selected design objects and selects the specified design object. Default.
- **@add (-Add)** — Adds the specified design object to the current selection set.

Return Values

VOID.

Description

If you omit either the name or type argument, this function selects all design objects that match the pattern of the argument specified.

To add the selected design object to the current selection set, you use the @add value; otherwise, this function selects the design object and unselects any previous selections.

You can press the Select key, as a shortcut method for executing the \$select_object() function, as shown in “[Key Names Mapped to Functions & Scopes](#)” on page 804.

Examples

1. This example selects the “logo” design object and adds it to the current selection set.

```
$select_object("logo", "Mgc_file", @add);
```

2. This example selects the same design object by using command syntax from the popup command line.

```
sel ob logo Mgc_file -add
```

3. This example uses a wildcard to select all design objects that begin with “config”.

```
sel ob config*
```

Related Topics

[\\$select_all\(\)](#)

[\\$select_by_name\(\)](#)

[\\$select_by_type\(\)](#)

[\\$select_config_entry\(\)](#)

[\\$select_reference\(\)](#)

[\\$select_tool\(\)](#)

[\\$select_trash_object\(\)](#)

[\\$select_version\(\)](#)

[\\$unselect_by_name\(\)](#)

[\\$unselect_by_type\(\)](#)

[\\$unselect_trash_object\(\)](#)

[A Design Object](#)

\$select_reference()

Scope: dmgr_reference_list_area

Prerequisite: To use this function, you must be in an active Reference window.

Selects the specified reference.

Usage

`$select_reference("to_name", "to_type", "to_version", mode)`

SElect REference to_name to_type to_version mode

Arguments

- **to_name**
A string that specifies the absolute pathname of the target design object. You can specify the pathname using UNIX System V wildcards.
- **to_type**
A string that specifies the type of the target design object. You can specify the type by using UNIX System V wildcards.
- **to_version**
An integer that specifies the version number of the target design object. The default is 0, which represents the current version.
- **mode**
A switch name that indicates whether other design objects remain selected. Choose one of the following:
 - **@reselect (-Reselect)** — Unselects all currently selected references and selects the specified reference. Default.
 - **@add (-Add)** — Adds the specified reference to the current selection set.

Return Values

VOID.

Description

You must specify the absolute pathname of the target design object for the `to_name` argument.

By default, this function selects the current version of the specified reference. To select a specific version of a reference, you must specify the `to_version` argument.

To add the specified reference to the current selection set, you use the `@add` value; otherwise, this function selects the specified reference and unselects any previous selections.

Examples

1. This example selects the reference to version 3 of the *saturn* target design object and unselects any previously selected references.

```
$select_reference( "$PROJ_XYZ/proj/example/saturn",  
  "Document_file", 3, @reselect);
```

2. This example selects the same reference by using command syntax from the popup command line.

```
sel re $PROJ_XYZ/proj/example/saturn Document_file 3 -reselect
```

Related Topics

[\\$add_reference\(\)](#)

[\\$show_references\(\)](#)

[\\$delete_reference\(\)](#)

[\\$unselect_reference\(\)](#)

[References](#)

\$select_tool()

Scope: dmgr_tool_area

Prerequisite: To use this function, you must be in an active Tools window.

Selects the specified tool.

Usage

`$select_tool("name", "type", mode)`

SElect TOol name *type* *mode*

Arguments

- **name**
A string that specifies the name of the tool. You can specify the name by using UNIX System V wildcards.
- **type**
A string that specifies the type of the tool. You can specify the type by using UNIX System V wildcards. The default is an empty string.
If you do not specify the type, this function selects the tool whose name matches the name you specified.
- **mode**
A switch name that indicates whether other tools remain selected. Choose one of the following:
 - **@reselect (-Reselect)** — Unselects all currently selected tools and selects the specified tool. Default.
 - **@add (-Add)** — Adds the specified tool to the current selection set.

Return Values

VOID.

Description

To add the specified tool to the current selection set, you use the @add value. Otherwise, this function selects the specified tool and unselects any previous selections.

Examples

1. This example selects the “dss” tool and adds it to the set of currently selected tools.

```
$select_tool("dss", "mgc_dss_tool", @add);
```


2. This example selects the same tool, by using command syntax from the popup command line.

```
sel to dss mgc_dss_tool -add
```

Related Topics

[\\$select_all\(\)](#)

[\\$select_config_entry\(\)](#)

[\\$select_version\(\)](#)

[\\$unselect_all\(\)](#)

[\\$unselect_tool\(\)](#)

[Invoking from the Tools Window](#)

\$select_toolbox()

Scope: dmgr_toolbox_area

Prerequisite: To use this function, you must be in an active Tools window that currently shows toolboxes.

Selects the specified toolbox.

Usage

`$select_toolbox("name", mode)`

SElect TOOLBox name *mode*

Arguments

- **name**

A string that specifies the absolute pathname of the toolbox. You can specify the name using UNIX System V wildcards.

- **mode**

A switch name that indicates whether other toolboxes remain selected. Choose one of the following:

- **@reselect (-Reselect)** — Unselects all currently selected toolboxes and then selects the specified toolbox. Default.
- **@add (-Add)** — Adds the specified toolbox to the current selection set.

Return Values

VOID.

Description

You must specify the absolute pathname of the toolbox. To add the specified toolbox to the current selection set, you use the @add value; otherwise, this function selects the specified toolbox and unselects any previous selections.

Examples

1. This example selects the `$PROJ_XYZ/proj/toolbox1` toolbox and adds it to the set of currently selected toolboxes.

```
$select_toolbox("$PROJ_XYZ/proj/toolbox1", @add);
```

2. This example selects the same toolbox, by using command syntax from the popup command line.

```
sel toolb $PROJ_XYZ/proj/toolbox1 -add
```

Related Topics

[`\$select_config_entry\(\)`](#)

[`\$select_version\(\)`](#)

[`\$select_tool\(\)`](#)

[`\$unselect_toolbox\(\)`](#)

[Toolboxes](#)

\$select_trash_object()

Scope: dmgr_trash_area

Prerequisite: To use this function, you must be in an active Trash window.

Selects the specified trash object.

Usage

`$select_trash_object("name", "type", mode)`

SELEct TRash Object name *type mode*

Arguments

- **name**

A string that specifies the name of the trash object to be selected. You can specify the name by using UNIX System V wildcards.

- **type**

A string that specifies the type of the trash object to be selected. You can specify the name using UNIX System V wildcards. If you use a wildcard to specify the type, this function selects the first trash object whose name matches the wildcard pattern you specified.

The default is an empty string.

- **mode**

A switch name that indicates whether other trash objects remain selected. Choose one of the following:

- **@reselect (-Reselect)** — Unselects all currently selected trash objects and selects the specified trash object. Default.
- **@add (-Add)** — Adds the specified trash object to the current selection set.

Return Values

VOID.

Description

To add the specified trash object to the current selection set, you use the @add value.

Otherwise, this function selects the specified trash object and unselects any previous selections.

Examples

This example selects the *new_version* trash object and adds it to the set of currently selected trash objects.

```
$select_trash_object("new_version", "dme_config_do", @add);
```

Related Topics

[`\$select_config_entry\(\)`](#)

[`\$unselect_trash_object\(\)`](#)

[`\$select_version\(\)`](#)

\$select_version()

Scope: dmgr_version_list_area

Prerequisite: To use this function, you must be in an active version window.

Selects the specified design object version.

Usage

`$select_version("name", "type", version, mode)`

SElect VErsion name type *version mode*

Arguments

- **name**
A string that specifies the leaf name of the design object.
- **version**
An integer that specifies the version number of the design object. The default is 0, which represents the current version.
- **type**
A string that specifies the type of the design object version.
- **mode**
A switch name that indicates whether other versions remain selected. Choose one of the following:
 - **@reselect (-Reselect)** — Unselects all currently selected design object versions and selects the specified version. Default.
 - **@add (-Add)** — Adds the specified design object version to the current selection set.

Return Values

VOID.

Description

By default, this function selects the current version of the design object. To select a specific version, you must specify the version argument.

To add the specified version to the current selection set, you use the @add value. Otherwise, this function selects the specified version and unselects any previous selections.

Examples

1. This example selects version 3 of the “new_hampshire” design object and unselects any previously selected versions.

```
$select_version("new_hampshire", "Document_file",3,@reselect);
```

2. This example selects the same version, by using command syntax from the popup command line.

```
sel ve new_hampshire Document_file 3 -reselect
```

Related Topics

[\\$select_config_entry\(\)](#)

[\\$select_tool\(\)](#)

[\\$select_reference\(\)](#)

[\\$unselect_version\(\)](#)

[Versions](#)

\$set_build_rules()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window and must select at least one primary configuration entry.

Sets the build rules for the selected primary entries.

Usage

```
$set_build_rules(ref_trav, [obj_name], [obj_name_flag], [obj_type], [obj_type_flag],  
                    [obj_prop_name], [obj_prop_value], [obj_prop_flag], [ref_prop_name], [ref_prop_value],  
                    [ref_prop_flag])
```

SET BUild Rules ref_trav obj_name obj_name_flag obj_type obj_type_flag obj_prop_name
obj_prop_value obj_prop_flag ref_prop_name ref_prop_value ref_prop_flag

Edit > Set Build Rules

Configuration window popup menu > **Set Build Rules**

Arguments

- **ref_trav**

A name that specifies whether referenced objects of the selected primary entries are added to the configuration during the build operation. Choose one of the following options:

@on — All referenced design objects of the selected primary entries are included in the configuration object during the build.

@off — Referenced design objects of the selected primary entries are not included in the configuration object during the build.

- **obj_name**

A vector of strings that specify the inclusion or exclusion filters for design objects based on their pathname. The format of this vector of strings is:

[“pattern1”, “pattern2”, ..., “patternN”]

Each pattern argument in this vector is a string. You can specify the pattern string by using UNIX System V regular expressions.

- **obj_name_flag**

A vector of strings each of which specify whether to include or exclude the design objects that match its corresponding pattern string in the obj_name argument. The format of this vector of strings is:

[“name_flag1”, “name_flag2”, ..., “name_flagN”]

Each name_flag string has one of two possible values: *exclude* or *include*. Each name_flag string corresponds to a obj_name argument based on its position in the vector. For example, the name_flag1 string in the vector above holds the first position in the vector and

corresponds to the `pattern1` string in the `obj_name` vector above. You can specify the `name_flag` string by using UNIX System V regular expressions.

- **obj_type**

A vector of strings that specify the inclusion or exclusion filters for design objects based on their type. The format of this vector of strings is:

[“type1”, “type2”, ..., “typeN”]

Each type argument in this vector is a string that specifies a design object type. You can specify the type string by using UNIX System V regular expressions.

- **obj_type_flag**

A vector of strings each of which specify whether to include or exclude the design objects that match its corresponding type string in the `obj_type` argument. The format of this vector of strings is:

[“type_flag1”, “type_flag2”, ..., “type_flagN”]

Each `type_flag` string has one of two possible values: *exclude* or *include*. Each `type_flag` string corresponds to a `obj_type` argument based on its position in the vector. For example, the `type_flag1` string in the vector above holds the first position in the vector and corresponds to the `pattern1` string in the `obj_type` vector. You can specify the pattern string by using UNIX System V regular expressions.

- **obj_prop_name**

A vector of strings that specify the inclusion or exclusion filters for design objects based on the property names that they hold. The format of this vector of strings is:

[“prop_name1”, “prop_name2”, ..., “prop_nameN”]

Each `prop_name` argument in this vector is a string that specifies the name of the property. You can specify the `prop_name` string by using UNIX System V regular expressions.

- **obj_prop_value**

A vector of strings that specify the inclusion or exclusion filters for design objects based on the property values that they hold. The format of this vector of strings is:

[“prop_value1”, “prop_value2”, ..., “prop_valueN”]

Each `prop_value` argument in this vector is a string that specifies the value of a property. You can specify the `prop_value` string by using UNIX System V regular expressions.

- **obj_prop_flag**

A vector of strings each of which specify whether to include or exclude the design objects that match both its corresponding string in the `obj_prop_name` argument and its corresponding string in the `obj_prop_value` argument. The format of this vector of strings is:

[“prop_flag1”, “prop_flag2”, ..., “prop_flagN”]

Each `prop_flag` string in this vector has one of two possible values: *exclude* or *include*. Each `prop_flag` string corresponds to a `obj_prop_name` and a `obj_prop_value` argument based on

its position in the vector. For example, the `prop_flag1` string in the vector above holds the first position in the vector and corresponds to both the `prop_name1` string in the `obj_prop_name` vector and the `prop_value1` string in the `obj_prop_value` vector.

- **ref_prop_name**

A vector of strings that specify the inclusion or exclusion filters for design objects based on the property names that their references hold. The format of this vector of strings is:

[“ref_prop_name1”, “ref_prop_name2”, ..., “ref_prop_nameN”]

Each `ref_prop_name` argument in this vector is a string that specifies the name of the property. You can specify the `ref_prop_name` string by using UNIX System V regular expressions.

- **ref_prop_value**

A vector of strings that specify the inclusion or exclusion filters for design objects based on the property values that their references hold. The format of this vector of strings is:

[“ref_prop_value1”, “ref_prop_value2”, ..., “ref_prop_valueN”]

Each `ref_prop_value` argument in this vector is a string that specifies the value of a property that their references hold. You can specify the `ref_prop_value` string by using UNIX System V regular expressions.

- **ref_prop_flag**

A vector of strings each of which specify whether to include or exclude the design objects that match both its corresponding string in the `ref_prop_name` argument and its corresponding string in the `ref_prop_value` argument. The format of this vector of strings is:

[“ref_prop_flag1”, “ref_prop_flag2”, ..., “ref_prop_flagN”]

Each `ref_prop_flag` string in this vector has one of two possible values: *exclude* or *include*. Each `ref_prop_flag` string corresponds to a `ref_prop_name` and a `ref_prop_value` argument based on its position in the vector. For example, the `ref_prop_flag1` string in the vector above holds the first position in the vector and corresponds to both the `ref_prop_name1` string in the `ref_prop_name` vector and the `ref_prop_value1` string in the `ref_prop_value` vector.

Return Values

VOID.

Description

The interactive function `$set_build_rules()` brings up the Set Build Rules dialog box, which enables you to set the build rules for the selected primary entries interactively. You can also execute this function from the popup command line. [Figure 5-1](#) shows the Set Build Rules dialog box. Setting the build rules for primary entries tells the Pyxis Project Manager application which design objects to include and to exclude from your configuration when you execute the `$build_configuration()` function.

When a single primary entry is selected, the Set Build Rules dialog box is displayed, showing the current build rules for the selected entry as default values. When multiple primary entries are selected, the Set Build Rules dialog box is displayed without any default values. When multiple primary entries are selected, all selected entries are set according to the values set in the Set Build Rules dialog box. Even though the `$set_build_rules()` function can be executed on multiple entries, each entry still has its own set of build rules.

The Set Build Rules dialog box enables you to set reference traversal to either on or off. If you set reference traversal to on, all referenced design objects of the selected primary entries are included in the configuration object during the build. The default setting for reference traversal is off.

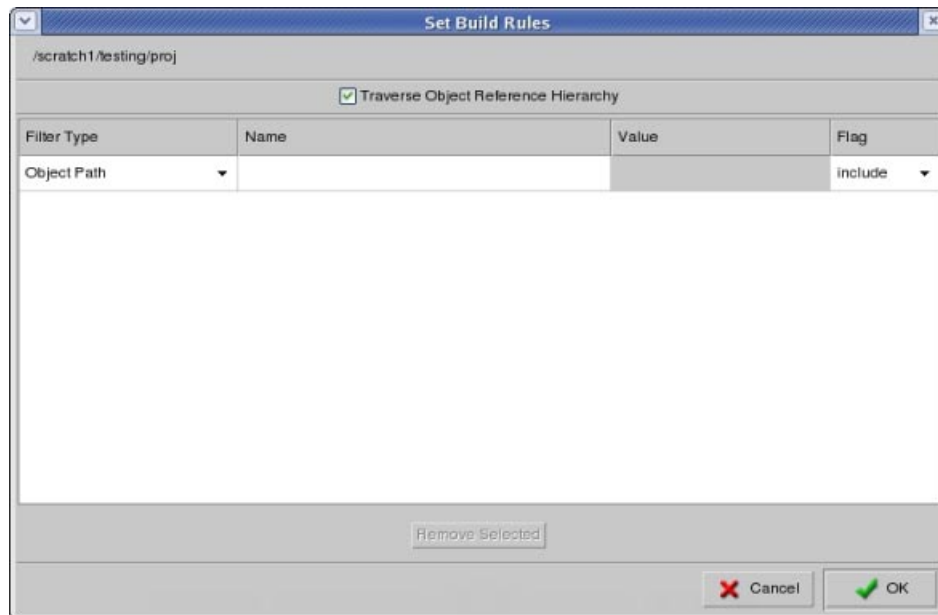
By default, containment traversal is always set to on, which means that a primary entry's contained objects are added to the configuration during a build. You can use the Set Build Rules dialog box to prevent specific design objects from being included in the configuration, by setting the following filters:

- **Object Path** — Filters design objects based on their pathname.
- **Object Type** — Filters design objects based on their type.
- **Object Property** — Filters design objects based on object properties held by the object. You can specify the property value as a null string (""), which then filters objects based solely on the property name.
- **Reference Property** — Filters design objects based on the properties held by the object's references. You can specify the property value as a null string (""), which then filters objects based solely on the property name.

If you do not specify build rules for a primary entry, it inherits the default build rules. The default build rules include all contained design objects and all referenced design objects.

[Figure 5-1](#) shows the Set Build Rules dialog box.

Figure 5-1. Set Build Rules Dialog Box



Examples

This example sets the build rules for a single selected primary entry using function syntax from the popup command line.

```
$set_build_rules(@on, ["^.*\\.lib$"], ["exclude"], ["^Image_file"],  
["exclude"], ["release_1.1"], ["", ["exclude"], ["release_1.1"],  
[""], ["exclude"]]);
```

The preceding example sets the following build rules:

- Reference traversal is set to *on*, which includes all referenced design objects in the configuration.
- The first filter is the object path filter, which uses a regular expression to exclude design objects that have an extension of *.lib*.
- The second filter is the object type filter, which is set to exclude all design objects of type *Image_file*.
- The third filter is the object property filter, which is set to exclude design objects that hold the property *release_1.1*. The null string denotes that the property value is not checked.
- The fourth filter is the reference property filter, which is set to exclude design objects whose references hold the property *release_1.1*. The null string denotes that the reference property values are not checked.

Related Topics

[`\$build_configuration\(\)`](#)

\$set_file_compilation_options()

Scope: lang_ifc

Prerequisite: To use this function, you must select a language file in the Project Navigator window.

Sets the compilation options for the indicated HDL file.

Usage

`$set_file_compilation_options("hdl_obj_path", "hdl_type", use_default_options, compiler, "adms_flags", "questa_or_eldo_flags", includes)`

Edit > Language > Compilation Options

Arguments

- **hdl_obj_path**
A string value specifying the pathname, with no extension, of the target HDL file.
- **hdl_type**
A string value specifying the type of the target HDL file, such as "mgc_verilog".
- **use_default_options**
A Boolean value. If true, then the file compilations are no longer used to override the project settings. The remaining arguments to this builtin are ignored. If false, then the remaining arguments are used to set file specific compilation options, and those are used when the file is compiled.
- **compiler**
Not used for Verilog AMS or VHDL AMS. For the remaining types, this is the file-specific equivalent to the `verilog_compiler`, `verilog_a_compiler`, or `vhdl_compiler` arguments in the `$set_project_options()` command.
- **adms_flags**
Depending on the target HDL file's type, this is equivalent to the `verilog_adms_flags`, `verilog_ams_flags`, `verilog_a_adms_flags`, or `vhdl_adms_flags` arguments in the `$set_project_options()` command.
- **questa_or_eldo_flags**
Only relevant for Verilog, VHDL, or Verilog-A source files. Depending on the target HDL file's type, this is equivalent to the `verilog_questa_flags`, `vhdl_questa_flags`, or `verilog_a_eldo_flags` arguments in the `$set_project_options()` command.
- **includes**
Only relevant for Verilog, Verilog A, and Verilog AMS source files. These includes are equivalent to the arguments `verilog_includes`, `verilog_ams_includes`, or `verilog_a_includes` in the command `$set_project_options()`.

Return Values

VOID.

Description

If `use_default_options` is true, then file-specific options are not used, and the HDL is compiled according to the project-level compilation options.

Examples

The following example sets the compilation options for the specified *inv.v* HDL file. As “`use_default_options`” is set to false, “`verilog_adms_flags`” is set to “-nocheck” and “`verilog_questa_flags`” is set to “-compat”. These options are used when the file is compiled.

```
$set_file_compilation_options("$TESTDIR/test_flow/hdl_lib/inv_Verilog/inv", "mgc_verilog",  
@false, @questa, "-nocheck", "-compat", []);
```

Related Topics

[\\$set_project_options\(\)](#)

\$set_ini_mappings()

Scope: lang_ifc

Prerequisite: To use this function, the context must be the Project Navigator window.

Updates the ADMS INI file for the current hierarchy to include the given set of compiled ADMS libraries.

Usage

`$set_ini_mappings([ini_list])`

Edit > Language > Manage INI Mappings

Arguments

- **ini_list** vector format [["lib_name", "lib_path"], [...]]

This list of vectors could include existing, modified, and new mappings.

Return Values

VOID.

Examples

The following example adds a mapping from the current project's ini file to the ADMS library at */tmp/my_adms_models*, and removes any other mappings to external compiled libraries:

```
$set_ini_mappings([["MY_ADMS_MODELS", "/tmp/my_adms_models"]]);
```


\$\$set_location_map_entry()

Scope: dme_dn_tk

Overrides a location map entry in memory.

Usage

```
$$set_location_map_entry("soft_name", "hard_name")
```

Arguments

- **soft_name**
A string that specifies the soft prefix whose hard pathname is to be overwritten.
- **hard_name**
A string that specifies the new hard pathname to associate with the soft prefix.

Return Values

VOID.

Description

The toolkit function `$$set_location_map_entry()` replaces the location map entry which corresponds to the specified soft prefix. The new entry contains only the specified hard pathname. If the location map has not been read into memory, this function forces the map to be read first, before the entry is replaced.

This function replaces the location map entry by matching the soft prefix you specified with a soft prefix in the location map. If a matching soft prefix is found in the location map and its first corresponding hard pathname differs from the `hard_name` argument you specified, this function replaces the hard pathname with the `hard_name` string. If a matching soft prefix is found in the location map and its first corresponding hard pathname is the same as the `hard_name` argument you specified, this function does not perform the replacement.

If you execute the `$$set_location_map_entry()` function without specifying the `hard_name` argument, this function displays an error message stating that a hard pathname must be supplied. If a matching soft prefix does not exist in the location map, this function displays an error message stating that a matching soft prefix cannot be found.

Examples

This example replaces the first hard pathname associated with the `$DDMS_DOC` soft prefix in the in-memory location map, with the `/project/dm/src/docs` hard pathname.

```
$$set_location_map_entry("$DDMS_DOC", "/project/dm/src/docs");
```

The location map entry before executing the command:

Soft Name: `$DDMS_DOC`

Hard Name: `//ddms/local_user/ddms.proj/src/documents.hm`

//project/ddms/src/documents.hm

The location map entry after executing the command:

Soft Name: *\$DDMS_DOC*

Hard Name: */project/dm/src/docs*

Related Topics

[\\$change_location_map_entry\(\)](#)

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

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

[\\$\\$show_location_map\(\)](#)

\$\$set_monitor_flag()

Scope: dm_config_tk

Changes a single attribute of the configuration monitoring setup.

Usage

\$\$set_monitor_flag(index, value)

Arguments

- **index**

A name that specifies the configuration monitoring attribute that you want to change.

- **@noerrors** — Specifies whether error messages are filtered out of configuration monitor status reports. If you specify this attribute and you set the value argument to @true, error messages are not reported. If you specify this attribute and you set the value argument to @false, all error messages are reported.
- **@nowarnings** — Specifies whether warning messages are included in configuration monitor status reports. If you specify this attribute and you set the value argument to @true, warning messages are not reported. If you specify this attribute and you set the value argument to @false, all warning messages are reported.
- **@noinfo** — Specifies whether informational messages are included in configuration monitor status reports. If you specify this attribute and you set the value argument to @true, informational messages are not reported. If you specify this attribute and you set the value argument to @false, informational messages are reported.
- **@show_window** — Specifies whether the monitor window is automatically displayed during configuration operations. If you specify this attribute and you set the value argument to @true, the configuration monitor window is automatically displayed during configuration operations. If you specify this attribute and you set the value argument to @false, the configuration monitor window is not displayed during configuration operations.
- **@window_mode** — Specifies whether configuration monitoring status is output to the monitor window or to the transcript area. If you specify this attribute and you set the value argument to @true, configuration monitoring status is output to the monitor window only. If you specify this attribute and you set the value argument to @false, configuration monitoring status is output to the transcript area only.
- **@enable_clearing** — Specifies whether the configuration monitor window is automatically cleared upon each new configuration operation. If you specify this attribute and you set the value argument to @true, the configuration monitor window is cleared each time you execute a configuration operation. If you specify this attribute and you set the value argument to @false, the configuration monitor window is not cleared when you execute a new configuration operation. Instead, the

most recent status information is appended to the bottom of the information previously reported in the configuration monitor window.

- **value**

A Boolean that specifies the value of its associated index argument. This argument accepts a Boolean value of either @true or @false.

Return Values

VOID.

Description

The toolkit function `$$set_monitor_flag()` enables you to change individual configuration monitoring attributes. This function changes the same monitoring attributes that you specify using the `$$setup_monitor()` function. However, this function enables you to change a single monitoring attribute, instead of having to specify the values of those attributes that you do not want to change. You can use the `$$get_monitor_flag()` function to query the value of any of the configuration monitoring attributes.

Examples

1. This example specifies to report configuration status to the monitor window.

```
$$set_monitor_flag(@window_mode, @true);
```

2. This example specifies to automatically display the monitor window during configuration operations.

```
$$set_monitor_flag(@show_window, @true);
```

3. This example specifies not to clear the monitor window each time a configuration operation is executed. Instead, the most recent status information is appended to the bottom of the information previously reported in the monitor window.

```
$$set_monitor_flag(@enable_clearing, @false);
```

4. This example specifies to filter all warning messages from configuration monitor reports.

```
$$set_monitor_flag(@nowarnings, @true);
```

Related Topics

[\\$\\$get_monitor_flag\(\)](#)

[\\$\\$set_monitor_verbosity\(\)](#)

[\\$\\$get_monitor_verbosity\(\)](#)

[\\$setup_monitor\(\)](#)

\$\$set_monitor_verbosity()

Scope: dm_config_tk

Sets the configuration monitor verbosity level.

Usage

```
$$set_monitor_verbosity(ranking)
```

Arguments

- **ranking**

An integer with a value in the range of 0 to 5. The integer corresponds to the verbosity setting of the configuration monitor, where a value of 1 is terse, a value of 5 is verbose, and all values in between are gradations of verbosity between the two extremes. A value of 0 indicates no ranking is applied; messages with a ranking of 0 are always be printed regardless of the monitor verbosity setting.

Return Values

VOID.

Description

The toolkit function `$$set_monitor_verbosity()` sets the monitor verbosity level to the value of the ranking argument. You can use the `$$get_monitor_verbosity()` function to return the configuration monitor verbosity setting.

Examples

This example sets the monitoring verbosity level to 5, specifying the configuration monitor should report all available information, including errors, warnings, failures, and other information messages.

```
$$set_monitor_verbosity(5);
```

Related Topics

[\\$\\$get_monitor_verbosity\(\)](#)

[\\$\\$set_monitor_verbosity\(\)](#)

[\\$\\$set_monitor_flag\(\)](#)

[\\$setup_monitor\(\)](#)

\$set_next_tool_env()

Scope: dm_tool_tk

Adds an environment variable with the given value to the next tool environment.

Usage

```
$set_next_tool_env("name", "value")
```

Arguments

- **name**
A string representing the name of the environment variable to add.
- **value**
A string representing the value of the environment variable.

Return Values

VOID.

Description

The value is read literally, and you are unable to dynamically append to the current value of the environment variable, if it already exists. If the environment variable already exists in the current environment, it is replaced with the specified value.

The last call to `$set_next_tool_env()` or `$unset_next_tool_env()` takes precedence.

Examples

```
$set_next_tool_env("A", "buzz")
```

Related Topics

[\\$get_next_tool_env\(\)](#)

[\\$unset_next_tool_env\(\)](#)

\$\$set_object_path_filter()

Scope: dm_config_tk

Sets the value of the object path filter for the specified primary entry.

Usage

\$\$set_object_path_filter([criteria], "name", "type", *version*)

Arguments

- **criteria**

A vector of string vectors that sets the object path filter. The format of the vector of string vectors is:

```
[[“path pattern1”, “path pattern2”, “path patternN”], [“exclude1”, “include2”,  
“excludeN”]]
```

The path pattern is a string that specifies the pathname of a design object. The include or exclude string determines if a design object is included or excluded from the configuration. You can use UNIX System V regular expressions to specify the path pattern.

- **name**

A string that specifies the pathname of the primary entry.

- **type**

A string that specifies the type of the primary entry.

- **version**

An integer that specifies the version number of the primary entry. The default is 0, which represents the current version.

Return Values

VOID.

Description

Setting this filter prior to executing the \$\$build_configuration() function enables you to either include or exclude design objects from the configuration, based on their pathname.

There is no limit on the number of elements in each subvector of the criteria vector. However, each pathname in the first subvector must have a corresponding include or exclude string in the second subvector.

To return the object path filter value of a primary entry, use the \$\$get_object_path_filter() function. To reset each of the primary entry's filters, use the \$\$clear_entry_filter() function.

Examples

1. This example opens the configuration *config_x* and sets the object path filter for the primary entry *proj*.

```
$$open_configuration("$PROJECT_XYZ/d1/config_x", 0, @read_write);  
$$set_object_path_filter( ["^$PROJECT_XYZ/proj/new.file$",  
    "$PROJECT_XYZ/proj/old.f ile$", ["exclude", "exclude"]],  
    "$PROJECT_XYZ/proj", "Mgc_container", 0);
```

The preceding example sets the object path filter so that it excludes only those design objects whose pathname matches *^\$PROJECT_XYZ/proj/new.file\$* or *^\$PROJECT_XYZ/proj/old.file\$*.

2. This example opens the same configuration and sets the object path filter to exclude the same design objects by using UNIX System V regular expressions.

```
$$open_configuration("$PROJECT_XYZ/d1/config_x", 0, @read_write);  
$$set_object_path_filter(["^.*.file$"], ["exclude"],  
    "$PROJECT_XYZ/proj", "Mgc_container", 0);
```

Related Topics

[\\$\\$build_configuration\(\)](#)

[\\$\\$set_object_type_filter\(\)](#)

[\\$\\$clear_entry_filter\(\)](#)

[\\$\\$set_reference_property_filter\(\)](#)

[\\$\\$get_object_path_filter\(\)](#)

[\\$\\$set_reference_traversal\(\)](#)

[\\$\\$set_object_property_filter\(\)](#)

\$\$set_object_property()

Scope: dme_do_tk

Sets the specified property on the specified design object.

Usage

`$$set_object_property("obj_name", "obj_type", obj_version, "prop_name", "prop_value")`

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **obj_type**
A string that specifies the type of the design object.
- **prop_name**
A string that specifies the name of the property. This string must not contain any blank spaces.
- **prop_value**
A string that specifies the value of the property.
- ***obj_version***
An integer that specifies the version number of the design object. The default is the current version.

Return Values

VOID.

Description

An object property consists of a property name and value, both of which are strings. As a design object evolves, its object properties carry forward to the next version, unless you specify the `object_version` argument. Adding an object property to a previous version has no effect on other versions of the design object.

If you omit the `object_version` argument, this function adds the property on the current version of the design object. To add a property to a previous version, you must specify the `object_version` argument.

If the property you specify in the `prop_name` argument does not currently exist, this function adds the object property to the design object. If the object property does exist, this function modifies it accordingly.

To add or modify an object property, you must have permission to edit the specified design object. Use the `$change_protection()` function to change the protection status of design objects.

Examples

This example adds the “QA_eng” property with the “mary” value to version 2 of the “schedule” design object.

```
$$set_object_property("$PROJECT_XYZ/d1/project/schedule",  
    "Document_file", 2, "QA_eng", "Mary);
```

Related Topics

[\\$\\$delete_object_property\(\)](#)

[\\$\\$get_object_property_value\(\)](#)

[\\$\\$get_object_properties\(\)](#)

[\\$\\$has_object_property\(\)](#)

[\\$change_protection\(\)](#)

\$\$set_object_property_filter()

Scope: dm_config_tk

Sets the value of the object property filter for the specified primary entry.

Usage

`$$set_object_property_filter([criteria], "name", "type", version)`

Arguments

- **criteria**

A vector of string vectors that sets the object property filter. The format of the vector of string vectors is:

```
[[["property_name1", "property_name2", "property_nameN"], ["value1",  
  "value2", "valueN"], ["exclude1", "include2", "excludeN"]]
```

The `property_name` is a string that specifies the name of the property. The value is a string that specifies the value of the property. The include or exclude string determines if a design object is included or excluded from the configuration.

- **name**

A string that specifies the pathname of the primary entry.

- **type**

A string that specifies the type of the primary entry.

- **version**

An integer that specifies the version number of the primary entry. The default is 0, which represents the current version.

Return Values

VOID.

Description

Setting this filter prior to executing the `$$build_configuration()` function enables you to include or exclude design objects from a configuration based on the design object's properties. If you omit the property value, this filter only checks the property name during a build.

There is no limit on the number of elements in each subvector or the criteria vector. However, each property name in the first subvector must have a corresponding property value in the second subvector and either an include or exclude string in the third subvector.

To return the object property filter value of a primary entry, you use the `$$get_object_property_filter()` function. To reset each of the primary entry's filters, you use the `$$clear_entry_filter()` function.

Examples

This example opens the “config_x” configuration and sets the object property filter for the “proj” primary entry.

```
$$open_configuration("$PROJECT_XYZ/d1/config_x", 0, @read_write);  
$$set_object_property_filter(["Release", "Engineer"],  
    ["alpha", "John Smith"], ["exclude", "include"],  
    "$PROJECT_XYZ/d1/proj", "Mgc_container", 0);
```

The preceding example sets the object property filter so that it excludes design objects with the “Release” object property and the “alpha” value, and includes those objects with the “Engineer” property and the “John Smith” value.

Related Topics

[\\$\\$build_configuration\(\)](#)

[\\$\\$set_object_type_filter\(\)](#)

[\\$\\$clear_entry_filter\(\)](#)

[\\$\\$set_reference_property_filter\(\)](#)

[\\$\\$get_object_property_filter\(\)](#)

[\\$\\$set_reference_traversal\(\)](#)

[\\$\\$set_object_path_filter\(\)](#)

\$\$set_object_type_filter()

Scope: dm_config_tk

Sets the value of the object type filter for the specified primary entry.

Usage

\$\$set_object_type_filter([criteria], "name", "type", *version*)

Arguments

- **criteria**

A vector of string vectors that sets the object type filter. The format of the vector of string vectors is:

```
[[“type pattern1”, “type pattern2”, “type patternN”], [“include”, “exclude”,  
“includeN”]]
```

The type pattern is a string that specifies the type of the design object. The include or exclude string determines if the design object is included or excluded from the configuration. You can use UNIX System V regular expressions to specify the type pattern.

- **name**

A string that specifies the pathname of the primary entry.

- **type**

A string that specifies the type of the primary entry.

- **version**

An integer that specifies the version number of the primary entry. The default is 0, which represents the current version.

Return Values

VOID.

Description

Setting this filter prior to executing the \$\$build_configuration() function enables you to include or exclude design objects from a configuration based on the design object's type.

There is no limit on the number of elements in each subvector of the criteria vector. However, each type in the first subvector must have a corresponding include or exclude string in the second subvector.

To return the object type filter value of a primary entry, you use the \$\$get_object_type_filter() function. To reset each of the primary entry's filters, you use the \$\$clear_entry_filter() function.

Examples

1. This example opens the “config_x” configuration and sets the object type filter for the “proj” primary entry.

```
$$open_configuration("$PROJECT_XYZ/d1/config_x", 0,  
    @read_write);  
  
$$set_object_type_filter(["Image.file", "Mgc.file",  
    ["include", "include"]], "$PROJECT_XYZ/d1/proj",  
    "Mgc_container", 0);
```

The preceding example sets the type filter so that it only includes design objects of type *Mgc.file* and *Image.file*.

2. This example uses UNIX System V regular expressions to set the object type filter to exclude all design objects with a *.file* extension.

```
$$open_configuration("$PROJECT_XYZ/d1/config_x", 0,  
    @read_write);  
  
$$set_object_type_filter(["^.*\\.file$"], ["exclude"]);
```

Related Topics

[\\$\\$build_configuration\(\)](#)

[\\$\\$clear_entry_filter\(\)](#)

[\\$\\$get_object_type_filter\(\)](#)

[\\$\\$set_object_path_filter\(\)](#)

[\\$\\$set_object_property_filter\(\)](#)

[\\$\\$set_reference_property_filter\(\)](#)

[\\$\\$set_reference_traversal\(\)](#)

\$set_project_options()

Scope: lang_ifc

Prerequisite: A project or external library must be selected in the Project Navigator window.

Sets the default compilation options for the currently open hierarchy.

Usage

```
$set_project_options(verilog_compiler, "verilog_questa_flags", "verilog_adms_flags",  
    verilog_includes, "verilog_ams_flags", verilog_ams_includes, verilog_a_compiler,  
    "verilog_a_adms_flags", "verilog_a_eldo_flags", verilog_a_includes, vhdl_compiler,  
    "vhdl_questa_flags", "vhdl_adms_flags", "vhdl_ams_flags")
```

Edit > Language > Compilation Options

Arguments

- **verilog_compiler, verilog_a_compiler and vhdl_compiler**

All of these source types have a choice of compiler. If Verilog or VHDL is compiled in ADMS, it is compiled with the -ms flag for compatibility with Questa simulators.

- **verilog_questa_flags and vhdl_questa_flags**

Options passed to the Questa compilers, or, if ADMS compilation is specified, options passed to the ADMS compiler with the "-ms" flag. (Either way, the options are applied to Questa compilation).

- **verilog_adms_flags, verilog_ams_flags, vhdl_adms_flags, vhdl_ams_flags**

Options passed to the ADMS compiler when compiling Verilog, Verilog AMS, VHDL, and VHDL AMS, respectively.

- **verilog_a_adms_flags**

If the Verilog A compiler is ADMS, verilog_a_adms_flags are the options passed to the valog compiler during Verilog A compilation.

- **verilog_a_eldo_flags**

The options passed to the vlac compiler when compiling Verilog A with Eldo.

- **verilog_includes, verilog_ams_includes, verilog_a_includes**

Each of these is a vector of strings, specifying include paths. Adding include paths to one of vectors is equivalent to using the "+incdir" flag in vlog or valog compilation, or the -I flag in vlac compilation.

Return Values

VOID.

Description

For any HDL files inside this project or external library, if the compilation options have not been overridden at the file level, the options specified here are used.

Examples

This example sets the project compilation options for the currently open hierarchy. The settings given below have the following effect on compilation:

- Verilog is compiled with the valog compiler for both ADMS and Questa simulation, with */tmp/incdir* passed to the compiler as an include directory.
- Verilog AMS compilation has no extra options and no include path.
- Verilog A is compiled with ADMS, with the **-analog** option, and no includes.
- VHDL is compiled with vlog for Questa simulation only, and has no extra options.
- VHDL AMS compilation has no extra options.

```
$set_project_options(@adms, "", "", ["/tmp/incdir"], "", [], @adms, "-analog", "", [], @questa,  
"", "", "");
```

Related Topics

[\\$set_file_compilation_options\(\)](#)

[\\$set_project_registration_options\(\)](#)

\$set_project_refresh_heartbeat()

Scope: dmgr_session_window

Schedules the Project Navigator window to automatically refresh every X minutes.

Usage

`$set_project_refresh_heartbeat(interval)`

SET PProject Refresh heartbeat

Setup > Preferences > Project Navigator panel

Arguments

- **interval**

The time, in minutes, that should pass between automatic refreshes. This must be a value between 0 and 20. Passing a value of 0 disables automatic refresh.

Return Values

VOID.

Description

A `$refresh_all()` call is scheduled every X minutes and executed immediately following any active user commands.

Related Topics

[\\$refresh_all\(\)](#)

\$set_project_registration_options()

Scope: lang_ifc

Prerequisite: To use this function, you must select a project or external library in the Project Navigator window.

Sets the registration options for the currently open hierarchy; these options are used for all Model Registration commands.

Usage

```
$set_project_registration_options([open_ports], [open_pins], pin_spacing, "shape");
```

Edit > Language > Registration Options

Arguments

- **open_ports**

A vector of string values. Lists port names which, if they appear on the model, are not added to the generated symbol. Unless the symbol already exists and has pins with matching names, the model ports are marked as open in the NCF file.

- **open_pins**

List of pins that should always appear on the generated symbol.

The type of this argument is a vector of vector values. Each of these vectors describes a pin to include on the generated symbol, in the format [{"name", implicit, "direction", "signal type", "pin type"}, ...].

The implicit value here is a Boolean. The direction, signal type, and pin type values are ignored if the value of implicit is true.

- **pin_spacing**

An integer value used when generating symbols.

- **Shape**

A string value used when generating symbols. Default value is "Box". Accepted shape values are: "And Gate", "Or Gate", "Xor Gate", "Buffer", "Box", "AndOr", "OrAnd", "Trapezoid", and "Adder".

Return Values

VOID.

Description

If the model does not have ports with matching names, the symbol pins are marked as open in the NCF file.

Implicit pins do not require any additional information other than the pin name. Implicits are mapped to the corresponding symbol properties, (unless the symbol has explicit pins with names that match those model ports).

Examples

The following example sets the default hierarchy options to include two additional pins (called "VDD" and "GND") on all generated symbols. For any registration created in this hierarchy, if these pins do not exist on the model, they are mapped as open pins in the registration entry.

```
$set_project_registration_options([], [ ["VDD", @false, "IN", "", ""],  
["GND", @false, "IN", "", "" ]], 2,"Box");
```

Related Topics

[\\$set_project_options\(\)](#)

\$\$set_protection()

Scope: dme_do_tk

Sets the protection of the specified design object.

Usage

```
$$set_protection("obj_name", "type", "protection")
```

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **type**
A string that specifies the type of the design object.
- **protection**
A string that specifies the new protection status of the design object.

Return Values

VOID.

Description

This function, which affects *all* versions of the design object and each fileset member, uses UNIX System V syntax. UNIX specifies three types of users: the *user* (u), or owner of the object; the *group* (g), or the user's group; and *others* (o), all others. The three permission settings are "r" for read, "w" for write, and "x" for execute. For example:

```
"u+w"
```

In the preceding example, the user adds write permission to the file by using the plus (+) sign. You can remove any permission by using a minus (-) sign, with the exception of "u-r". Another example:

```
"rwxr-x--"
```

Here, UNIX uses the "rwx" notation for each type of user. The first "rwx" refers to the user, the second refers to the user's group, and the third refers to everyone else. In this example, the user can read, write, and execute this file; the user's group can read and execute only; and all others cannot read, write, or execute this file.

In addition, you can specify the protection as an absolute integer by executing the `$$set_protection_numeric()` function.

Examples

This example changes the protection status of the design object *project* to read, write, and execute for the user; and to read and execute only for the user's group. All others cannot read, write, or execute this file.

```
$$set_protection("$PROJ_XYZ/proj/project","Mgc_file","rwxr-x--");
```

Related Topics

[\\$\\$get_object_protection\(\)](#)

[\\$\\$is_write_protected\(\)](#)

[\\$\\$is_read_protected\(\)](#)

[\\$\\$set_protection_numeric\(\)](#)

[\\$\\$is_writable\(\)](#)

\$\$set_protection_numeric()

Scope: dme_do_tk

Sets the protection of the specified design object by using an integer.

Usage

```
$$set_protection_numeric("obj_name", "type", protection)
```

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **type**
A string that specifies the type of the design object.
- **protection**
An integer that specifies the protection of the design object.

Return Values

VOID.

Description

The toolkit function `$$set_protection_numeric()` changes the protection status of the specified design object by using an absolute integer format. This function affects *all* versions of the design object and its fileset members.

The read, write, and execute permissions for *user* are as follows:

```
0o|4|0|0 // read permission
```

```
0o|2|0|0 // write permission
```

```
0o|1|0|0 // execute permission
```

In addition, you can specify the protection status as a string by using the `$$set_protection()` function.

Examples

This example uses an octal value and gives read permission to the user for the design object *chart*.

```
$$set_protection_numeric("$PROJECT_XYZ/d1/chart", "Mgc_file", 0o400);
```

Related Topics

[\\$\\$get_object_protection\(\)](#)

[\\$\\$is_write_protected\(\)](#)

[\\$\\$is_read_protected\(\)](#)

[\\$\\$set_protection\(\)](#)

\$\$is_writable()

\$\$set_reference_property()

Scope: dme_do_tk

Sets the specified property of the specified reference.

Usage

```
$$set_reference_property("obj_name", "obj_type", obj_version, "target_obj_name",  
    "target_type", target_version, "prop_name", "prop_value")
```

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **obj_type**
A string that specifies the type of the design object.
- **target_obj_name**
A string that specifies the pathname of the target design object.
- **target_type**
A string that specifies the type of the target design object.
- **prop_name**
A string that specifies the name of the property. This string must not contain any blank spaces.
- **prop_value**
A string that specifies the value of the property.
- ***obj_version***
An integer that specifies the version number of the source design object. The default is the current version.
- ***target_version***
An integer that specifies the version number of the target design object. The *target_version* also represents the state of the reference. If the *target_version* is greater than 0, the state of the reference this function operates on is *fixed*. If omitted or specified as 0, the state of the reference this function operates on is *current*.

The default is the current version.

Return Values

VOID.

Description

If the property you specify in the `prop_name` argument does not currently exist on the reference, this function adds the property to that reference. If the property does exist, this function modifies it accordingly.

This function can only add properties to references that hold the “creating_tool” property, whose value is “project manager”. When you add a reference by using the `$$add_reference()` function, the “creating_tool” property is automatically added to the reference.

To set a property on a previous version of the source design object, you must specify the `object_version` argument. Setting a reference property on a previous version has no effect on other versions of the source design object.

As other tools can create duplicate references, you can use the `$$set_reference_property_handle()` function for more accuracy when adding reference properties. To find a reference's target handle, you use the `$$get_object_references()` function.

If you specify the `target_version` argument, this function adds the property to all of the fixed references found whose name, type, property, and version number match the specified arguments. Because the `target_version` argument is specified, the state of the reference is fixed and the version number is checked.

If you omit the `target_version` argument, this function adds the property to all of the current references found whose name, type, and property match the specified arguments. Because the `target_version` argument is not specified, the state of the reference is current and the version number is not checked.

To add or modify a property, you must have permission to edit the specified design object. You can use the `$change_protection()` function to change the protection status of design objects.

Examples

1. This example adds the “status” property with the “complete” value to the reference from the “saturn” source design object to version 2 of the “ring” target design object. As the `target_version` argument *is* specified, the reference is fixed and the version number is checked.

```
$$set_reference_property(  
    "$PROJECT_XYZ/d1/project/saturn", "Document_file", ,  
    "$PROJECT_XYZ/d1/project/saturn/ ring", "Image_file", 2,  
    "status", "complete");
```

2. This example adds the “status” property with the “complete” value to the reference from the “saturn” source design object to the “ring” target design object. As the `target_version` argument is *not* specified, the reference is current and the version number is not checked.

```
$$set_reference_property(
```

```
"$PROJECT_XYZ/d1/project/saturn", "Document_file", ,  
"$PROJECT_XYZ/d1/project/saturn/ ring", "Image_file", ,  
"status", "complete");
```

3. This example assumes that the current version of the “saturn” design object is 5. The reference property is added to version 3, which is a previous version. Adding the reference property to version 3 has no effect on earlier or later versions of the source design object.

```
$$set_reference_property(  
"$PROJECT_XYZ/d1/project/saturn", "Document_file", 3,  
"$PROJECT_XYZ/d1/project/saturn /ring", "Image_file", ,  
"status", "complete");
```

Related Topics

[\\$\\$delete_reference_property\(\)](#)

[\\$\\$get_reference_properties\(\)](#)

[\\$change_protection\(\)](#)

[\\$\\$has_reference_property\(\)](#)

[\\$\\$set_reference_property_handle\(\)](#)

[\\$\\$add_reference\(\)](#)

\$\$set_reference_property_filter()

Scope: dm_config_tk

Sets the value of the reference property filter for the specified primary entry.

Usage

\$\$set_reference_property_filter([criteria], "name", "type", *version*)

Arguments

- **criteria**

A vector of string vectors that sets the reference property filter. The format of the vector of string vectors is:

```
[[["property_name1", "property_name2", "property_nameN"],  
  ["value1", "value2", "valueN"], ["exclude", "include", "excludeN"]]
```

The `property_name` is a string that specifies the name of the reference property. The value is a string that specifies the value of the property. The include or exclude string determines if a design object is included or excluded from the configuration.

- **name**

A string that specifies the pathname of the primary entry.

- **type**

A string that specifies the type of the primary entry.

- **version**

An integer that specifies the version number of the primary entry. The default is 0, which is the current version.

Return Values

VOID.

Description

Setting this filter prior to executing the `$$build_configuration()` function enables you to include or exclude design objects from a configuration during a reference traversal, based on the design object's reference properties. If you omit the reference property value, this function only checks the reference property name during a build.

There is no limit on the number of elements in each subvector of the criteria vector. However, each `property_name` in the first subvector must have a corresponding value in the second subvector and either an include or exclude string in the third subvector.

To return the object reference property filter value of a primary entry, you use the `$$get_reference_property_filter()` function. To reset each of the primary entry's filters, use the `$$clear_entry_filter()` function.

Examples

This example opens the “config_x” configuration and sets the reference property filter for the “proj” primary entry.

```
$$open_configuration("$PROJECT_XYZ/d1/config_x",0, @read_write);  
$$set_reference_property_filter([["Release", "Engineer"], ["alpha",  
    "John Smith"], ["exclude", "include"]], "$PROJECT_XYZ/d1/proj",  
    "Mgc_container", 0);
```

The preceding example sets the reference property filter to exclude design objects with the “Release” reference property and the “alpha” value, and to include those objects with the “Engineer” reference property and the “John Smith” value.

Related Topics

[\\$\\$build_configuration\(\)](#)

[\\$\\$set_object_property_filter\(\)](#)

[\\$\\$clear_entry_filter\(\)](#)

[\\$\\$set_object_type_filter\(\)](#)

[\\$\\$get_reference_property_filter\(\)](#)

[\\$\\$set_reference_traversal\(\)](#)

[\\$\\$set_object_path_filter\(\)](#)

\$\$set_reference_property_handle()

Scope: dme_do_tk

Sets the property of the specified reference by using the target reference handle.

Usage

```
$$set_reference_property_handle("obj_name", "obj_type", obj_version, target_handle,  
    "prop_name", "prop_value")
```

Arguments

- **obj_name**
A string that specifies the pathname of the source design object.
- **obj_type**
A string that specifies the type of the source design object.
- **target_handle**
An integer that specifies the handle of the target design object.
- **prop_name**
A string that specifies the name of the property. This string must not contain any blank spaces.
- **prop_value**
A string that specifies the value of the property.
- **obj_version**
An integer that specifies the version number of the source design object. The default is the current version.

Return Values

VOID.

Description

Setting a reference property by using the target handle enables you to specify distinctly which reference property to set. To find the reference's target handle, you use the `$$get_object_references()` function.

If the reference property you specify in the `prop_name` argument does not currently exist on the reference, this function adds the property to that reference. If the reference property does exist, this function modifies it accordingly.

This function can only add properties to references that hold the "creating_tool" property, whose value is "project manager". When you add a reference by using the `$$add_reference()` function, the `creating_tool` property is automatically added to the reference.

To set a reference property on a previous version of the source design object, you must specify the `object_version` argument. Setting a reference property on a previous version has no effect on other versions of the source design object.

To add or modify a reference property, you must have permission to edit the specified design object. You can use the `$change_protection()` function to change the protection status of design objects.

Examples

This example adds the “status” property and the “complete” value to the current version of the “saturn” source design object , using the reference target handle 1.

```
$$set_reference_property_handle("$PROJECT_XYZ/d1/saturn",  
    "Document_file", , 1, "status", "complete");
```

Related Topics

[\\$\\$delete_reference_property\(\)](#)

[\\$\\$has_reference_property\(\)](#)

[\\$\\$get_reference_properties\(\)](#)

[\\$\\$set_reference_property\(\)](#)

[\\$\\$add_reference\(\)](#)

[\\$change_protection\(\)](#)

[\\$\\$get_object_references\(\)](#)

\$\$set_reference_traversal()

Scope: dm_config_tk

Sets the reference traversal depth of the specified primary entry.

Usage

`$$set_reference_traversal(depth, "name", "type", version)`

Arguments

- **depth**

A name that specifies the traversal depth. Choose one of the following:

- **@on** — Searches all levels of the reference hierarchy when a build is performed.
- **@off** — Does not search the reference hierarchy.

- **name**

A string that specifies the pathname of the primary entry.

- **type**

A string that specifies the type of the primary entry.

- ***version***

An integer that specifies the version number of the primary entry. The default is 0, which represents the current version.

Return Values

VOID.

Description

If reference traversal is set to @on, the Pyxis Project Manager application traverses the references of the primary entry and checks whether to include them in the configuration. If reference traversal is set to @off, references are not traversed and are not included in the configuration.

Configurations of "Dme_config_do" type add the "creating_tool" reference property, whose value equals "Dme_config", to the configuration object. This reference property is always present. It indicates that the reference was created by the configuration object and should not be changed or deleted by the Pyxis Project Manager application, or any other tool.

A second reference property "Dme_config_ignore" whose value is "value__void", is placed on certain references to prohibit the `$$build_configuration()` function from traversing references that are added by the configuration object, but do not correspond to actual configuration entries.

Examples

This example opens the “my_config” configuration and sets the reference traversal to @off for the “project” primary entry .

```
$$open_configuration("$PROJECT_XYZ/d1/my_config",0, @read_write);
```

```
$$set_reference_traversal(@off, "$PROJECT_XYZ/d3/project", "Mgc_container", 0);
```

Related Topics

[\\$\\$build_configuration\(\)](#)

[\\$\\$is_entry_fixed\(\)](#)

[\\$\\$get_reference_traversal\(\)](#)

[\\$\\$is_entry_primary\(\)](#)

\$set_subinvoke_mode()

Scope: dm_tool_tk

Sets the tool invocation method to either send transcript messages to a new terminal window or to save transcript messages to a log file.

Usage

```
$set_subinvoke_mode("mode")
```

Arguments

- **mode**
 - **@xterm** — Transcript messages are sent to a new xterm window.
 - **@file** — Transcript messages are sent to a log file.

Return Values

VOID.

Description

The log file produced is available at *\$MGC_HOME/tmp/<tool_name_pid.txt>* whenever a tool is invoked.

Log files are produced in either a directory specified by the *\$MGC_TRANSCRIPT_DIR* environment variable or, if it is not set, *\$MGC_HOME/tmp/* by default. The log files follow the naming convention of *<tool_name>_<invocation_timestamp>.txt*.

Examples

```
$set_sub_invoke_mode(@xterm);
```

```
$set_sub_invoke_mode(@file);
```

Related Topics

[\\$get_subinvoke_mode\(\)](#)

\$\$set_target_path()

Scope: dm_config_tk

Sets the target path of the specified configuration entries.

Usage

\$\$set_target_path("target", "name", "type", *version*)

Arguments

- **target**
A string that specifies the target location of the specified configuration entries during a configuration operation. You can specify a relative or absolute pathname. If you specify a relative pathname, the final target pathname is the concatenation of the configuration operation's destination and the relative pathname.
- **name**
A string that specifies the pathname of the configuration entry.
- **type**
A string that specifies the type of the configuration entry.
- **version**
An integer that specifies the version number of the configuration entry. The default is 0, which represents the current version.

Return Values

VOID.

Description

The target path is a pathname that you can specify for retargetable entries. A configuration entry is retargetable if its parent container is not part of the configuration. When you perform either a release or copy of the configuration, the primary entry is then released or copied to its target path, rather than to the target directory of the release or copy operation.

To check if an entry is retargetable, you use the \$\$is_entry_retargetable() function. To return the current target path, you use the \$\$get_target_path() function.

If you specify a target pathname using an absolute pathname, the target pathname takes precedence over the release or copy destination. If you specify a target pathname using a relative pathname, the target path is concatenated to the end of the destination pathname of the release or copy. If you specify a target pathname as the empty string "", the entry's target path is the default which is the leaf name at the target destination of the release or copy.

Examples

This example opens the "config_a" configuration and sets the target path for the "file" configuration entry to *\$PROJECT_XYZ/project/release_dir2*.

```
$$open_configuration("$PROJECT_XYZ/design/config_a", 0,  
    @read_write);
```

```
$$set_target_path("$PROJECT_XYZ/project/release_dir2",  
    "$PROJECT_XYZ/d2/file", "Mgc_file", 0);
```

Related Topics

[\\$\\$build_configuration\(\)](#)

[\\$\\$is_entry_retargetable\(\)](#)

[\\$\\$get_target_path\(\)](#)

\$set_target_path()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window and must select at least one configuration entry.

Sets the target path of selected configuration entries in the active Configuration window.

Usage

`$set_target_path("pathname")`

SET TArget Path pathname

Edit > Set Target Path

Configuration window popup menu > **Set Target Path**

Arguments

- **pathname**

A string that specifies the target location of the selected configuration entries during a configuration operation. You can specify either a relative or an absolute pathname. If you specify a relative pathname, the final target pathname is the concatenation of the configuration operation's destination and the relative pathname. The default is the empty string ""

Return Values

VOID.

Description

When you perform either a release or copy of the configuration, the specified configuration entries are released or copied to their target path, not to the target directory of the release or copy operation.

The target path is a pathname that you can specify for retargetable configuration entries. A configuration entry is retargetable if its parent container is not part of the configuration. To check if an entry is retargetable, you use the `$$is_entry_retactable()` function. To return the current target path, you use the `$$get_target_path()` function.

If you specify a target pathname using an absolute pathname, the target pathname takes precedence over the release or copy destination. If you specify a target pathname using a relative pathname, the target path is concatenated to the end of the destination pathname of the release or copy. If you specify a target pathname as the empty string "", the entry's target path is the default which is the leaf name at the target destination of the release or copy.

The target pathname is interpreted differently, depending on whether you have a single or multiple configuration entries selected. If you execute this function with a single configuration entry selected, the specified target path is assumed to be the entire target path. If you execute

this function with multiple configuration entries selected, the specified target path is assumed to be a directory to which the leaf name of each entry is appended.

Examples

1. This example uses an absolute pathname to set the target path of the selected configuration entries to *\$PROJECT_XYZ/project/test_directory*.
2. This example uses a relative pathname to set the target path of the selected configuration entry to *staff*.

```
$set_target_path("$PROJECT_XYZ/project/test_directory");
```

```
$set_target_path("staff");
```

If the destination directory of the configuration operation is *\$PROJ_XYZ/project/test_directory*, the resulting target path for this example is *\$PROJ_XYZ/project/test_directory/staff*.

3. This example sets the target path by using command syntax from the popup command line.

```
set ta p $PROJECT_XYZ/project/test_directory
```

```
set ta p staff
```

Related Topics

[\\$build_configuration\(\)](#)

[\\$copy_configuration\(\)](#)

[\\$\\$get_target_path\(\)](#)

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

[\\$\\$is_entry_retargetable\(\)](#)

[Setting Target Paths](#)

\$set_technology()

Scope: dm_proj_tk

Sets the technology for the specified root HDO (project or external library) to the specified technology configuration.

Usage

```
$set_technology("root_hdo_path", "tech_config_path");
```

Arguments

- **root_hdo_path**
A string value containing the path to the project or external library to which the setting should be applied.
- **tech_config_path**
A string value containing the path to the technology configuration that should be used.

Return Values

VOID.

Examples

The following example sets the technology for the project at */tmp/project* to */tmp/generic_kit/configurations/generic_65n_6*:

```
$set_technology("/tmp/project", "/tmp/generic_kit/configurations");
```

Related Topics

[\\$create_dm_tech_lib\(\)](#)

[\\$get_technology\(\)](#)

[\\$create_tech_config_object\(\)](#)

\$set_toolbox_search_path()

Scope: dmgr_tk

Prerequisite: You can use this function from any Pyxis Project Manager window.

Sets the order of the toolbox search path to the specified pathnames.

Usage

```
$set_toolbox_search_path([pathname])
```

Arguments

- **pathname**

A vector of strings that defines the toolbox search path. You must specify absolute pathnames.

Return Values

VOID.

Description

Each vector in the toolbox search path represents a toolbox containing tool viewpoints. The Pyxis Project Manager application searches for tool viewpoints in order of path specification: the first toolbox specified is searched first, the second toolbox specified is searched second, until all toolboxes are searched.

Each toolbox pathname must begin with “/” or “//”. You cannot use relative pathnames to specify toolbox pathnames. You can specify hard or soft pathnames to specify toolbox pathnames.

This function sets the entire toolbox search path in one operation; it cannot add or delete an individual toolbox. To add toolboxes, you use the \$add_toolbox() function. To delete toolboxes, use the \$remove_toolbox() function.

Examples

This example sets the toolbox search path so that tool viewpoints in *\$PROJ_XYZ/proj/toolbox1* have precedence over tool viewpoints in the standard Mentor Graphics Tree toolbox. As a result, the Pyxis Project Manager application searches *toolbox1* prior to searching *\$MGC_HOME/toolbox*.

```
$set_toolbox_search_path(["$PROJ_XYZ/proj/toolbox1",  
    $MGC_HOME/toolbox"]);
```

Related Topics

[\\$add_toolbox\(\)](#)

[\\$save_toolbox_search_path\(\)](#)

[\\$get_toolbox_search_path\(\)](#)

[\\$view_toolboxes\(\)](#)

[\\$remove_toolbox\(\)](#)

[Toolbox Search Path Startup File](#)

\$\$set_version_depth()

Scope: dme_do_tk

Sets the version depth of the specified design object to the specified number.

Usage

```
$$set_version_depth("obj_name", "obj_type", version_depth)
```

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **obj_type**
A string that specifies the type of the design object.
- ***version_depth***
An integer that specifies the version depth of the design object. The default is the version depth for the object's type.

Return Values

VOID.

Description

This function attempts to delete versions that are beyond the new version depth, but has no effect on frozen versions. If you omit the *version_depth* argument or set the version depth to 0, this function sets the version depth to the design object's default version depth, as specified by its type.

To avoid completely the deletion of previous versions, set the version depth to -1, which enables the design object to have an infinite number of previous versions.

To determine the design object's current version depth, you use the `$$get_version_depth()` function. To change the version depth of a design object, you must have permission to edit the specified design object. You can use the `$change_protection()` function to change the protection status of design objects.

Examples

This example changes the version depth of the “manual” design object to 4.

```
$$set_version_depth("$PROJ_XYZ/proj/example/manual",  
    "Document_file", 4);
```

Related Topics

[\\$\\$delete_version\(\)](#)

[\\$\\$get_version_properties\(\)](#)

\$\$freeze_version()

\$\$revert_version()

\$\$get_object_versions()

\$\$unfreeze_version()

\$\$get_version_depth()

\$change_protection()

\$\$set_version_property()

Scope: dme_do_tk

Sets the specified property of the specified version.

Usage

`$$set_version_property("obj_name", "obj_type", version, "prop_name", "prop_value")`

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **obj_type**
A string that specifies the type of the design object.
- **prop_name**
A string that specifies the name of the property. This string must not contain any blank spaces.
- **prop_value**
A string that specifies the value of the property.
- ***version***
An integer that specifies the version number of the design object. The default is the current version.

Return Values

VOID.

Description

A version property consists of a name and a value, both of which are strings. Version properties always remain fixed to a particular version. As the design object evolves, the version property remains fixed to the version to which it was originally set.

If the property you specify in the `prop_name` argument does not currently exist, this function adds the property to that version. If the property does exist, this function modifies it accordingly.

By default, this function adds the property to the current version of the design object. To add a property to a previous version, you must specify the version argument. Adding a property to a previous version has no effect on other versions of the design object.

To add or modify a property, you must have permission to edit the specified design object. You can use the `$change_protection()` function to change the protection status of design objects.

Examples

This example adds the “status” property with the “complete” value to version 2 of the “saturn” design object.

```
$$set_version_property("$PROJECT_XYZ/d1/project/saturn",  
    "Image_file", 2, "status", "complete");
```

Related Topics

[\\$\\$delete_version_property\(\)](#)

[\\$\\$get_version_properties\(\)](#)

[\\$change_protection\(\)](#)

\$\$set_working_directory()

Scope: `dme_dn_tk`

Changes the value of the MGC working directory.

Usage

```
$$set_working_directory("new_directory")
```

Arguments

- **new_directory**

A string that contains any pathname. You should specify a hard pathname, if possible.

Return Values

A string containing the pathname of the MGC working directory.

Description

If you enter a relative name for the `new_directory` argument, the `$$set_working_directory()` function attempts to create an absolute pathname by checking the existing value of the MGC working directory if it has been set previously, or the value of the `MGC_WD` environment variable. If the environment variable is set, its value is appended to the beginning of the relative name and any existing `“/..”` or `“./”` string elements in the resulting pathname are eliminated.

If the `new_directory` argument is specified as an absolute pathname, any existing `“/..”` or `“./”` string elements in the resulting pathname are eliminated. If the pathname contains too many `“/..”` or `“./”` string elements, or if the pathname begins with a tilde (`~`), an error message states that you have entered an invalid pathname and the original working directory remains unchanged.

When the `new_directory` argument is successfully processed, the pathname is converted to a soft pathname and stored as the new working directory.

Examples

This example updates the MGC working directory to the absolute pathname `/usr/sandp/project/dir1`.

```
$$set_working_directory("/usr/sandp/project/dir1");
```

Related Topics

[\\$\\$fix_relative_path\(\)](#)

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

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

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

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

[\\$\\$is_relative_path\(\)](#)

[\\$get_navigator_directory_hard\(\)](#)

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

\$set_working_directory()

Scope: session_area

Prerequisite: You can use this function from any Pyxis Project Manager window.

Changes the value of the MGC working directory.

Usage

```
$set_working_directory("new_directory")
```

SET WORKing Directory

MGC > Location Map > Set Working Directory

Arguments

- **new_directory**

A string that contains the new working directory. You should specify a hard pathname, if possible.

Return Values

A string containing the pathname of the MGC working directory.

Description

If you enter a relative name for the new_directory argument, the \$set_working_directory() function attempts to create an absolute pathname by checking the existing value of the MGC working directory if it has been set previously, or the value of the MGC_WD environment variable. If the environment variable is set, its value is appended to the beginning of the relative name and any existing "../" or "./" string elements in the resulting pathname are eliminated.

If the new_directory argument is specified as an absolute pathname, any existing "../" or "./" string elements in the resulting pathname are eliminated. If the pathname contains too many "../" or "./" string elements, or if the pathname begins with a tilde (~), an error message states that you have entered an invalid pathname and the original working directory remains unchanged.

When the new_directory argument is successfully processed, the pathname is converted to a soft pathname and stored as the new working directory.

Examples

This example updates the MGC working directory to the absolute pathname *"/usr/sandp/project/dir1"*.

```
$set_working_directory("/usr/sandp/project/dir1");
```

Related Topics

[\\$\\$fix_relative_path\(\)](#)

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

\$\$get_hard_name()

\$\$is_relative_path()

\$get_navigator_directory()

\$set_working_directory()

\$get_navigator_directory_hard()

\$show_location_map()

\$\$get_soft_name()

\$setup_filter_active()

Scope: dmgr_model

Prerequisite: To use this function, a Navigator window must be activated.

Specifies filters that determine which objects are displayed in the active Navigator window.

Usage

`$setup_filter_active([name_wildcard], include_wildcard, [type_names], sort_by)`

SETUp Filter ACtive [name_wildcard] include_wildcard [type_names] sort_by

Navigator window popup menu > **Setup** > **Filter**

Arguments

- **name_wildcard**

A vector of strings that specifies which objects that are to be included or excluded from the navigator display. You can specify these strings using UNIX System V wildcards. If the wildcard field is empty, objects are filtered based only on their type. The format of this vector of strings is:

["string1", "string2", ..., "stringN"]

String is a string that specifies the leaf name of the object. The default is the last string(s) specified or the vector [*] if you have not executed this function in the current session.

- **include_wildcard**

A name that indicates whether object names that match one or more strings in the name_wildcard argument are included or excluded from the list of objects being displayed. The default is the last option specified or @include if you have not executed this function in the current session. Choose one of the following:

- **@exclude** — Excludes all objects which pass the name_wildcard filter.
- **@include** — Includes all objects which pass the name_wildcard filter.

- **type_names**

A vector of strings that specifies the design objects that are excluded based on their type. This argument defines the universe of design object types known to this function. If the type_names field is empty, all design object types are included and objects are filtered based only on their name. The format of this vector of strings is:

["string1", "string2", ..., "stringN"]

String is a string that specifies the type of the object. The default is the last type(s) specified or the vector [*] if you have not executed this function in the current session.

- **sort_by**

A name that specifies whether objects are displayed in a list sorted first by type and then by name, or if they are displayed in alphabetical order with no regard for type. The default is

the last option specified or @name if you have not executed this function in the current session. Choose one of the following:

- **@name** — Design objects are displayed in alphabetical order by design object name without any regard for type.
- **@type** — All design objects of a single type are displayed together in alphabetical order. The type groups are displayed alphabetically.

Return Values

VOID.

Description

The interactive function `$setup_filter_active()` specifies filters, based on a design object's name and type, that determine which design objects are displayed in the current Navigator window. The `type_names` argument defines the set of design objects that the `name_wildcard` and `include_wildcard` arguments operate on. That is, if you specify to include or exclude all design objects whose names match a specific pattern, this function only matches against those design objects which have successfully passed through the filters specified by the `type_names` argument. By default, no types are filtered out and navigator filters are set to include "*" which means that all files, except those that begin with a ".", are displayed.

You use the `sort_by` argument to specify how the design objects are displayed in the Navigator window. If you specify `@name`, the design objects are displayed alphabetically according to their name, without any regard for type. If you specify `@type`, the design object types are displayed alphabetically with all design objects of each type displayed alphabetically beneath each type.

This function sets navigation filters on the currently active Navigator window. You can use this filter as a specific override to the navigation filters that have been set by the `$setup_filter_all()` function. If the navigator is in reference mode, that is, if you are exploring references, the filters do not apply and the navigator display does not change.

The filters set by the `$setup_filter_active()` function are not saved and they do not change the filter settings in the Pyxis Project Manager default startup file. This function sets navigator filters for the active Navigator window for the current session only.

Examples

1. This example filters out all objects in the current Navigator window that are of some type other than "Mgc_container" or "Mgc_file" and do not have a "." extension. The objects are displayed in the Navigator window alphabetically.

```
$setup_filter_active([".*"], @include, ["Mgc_container",  
"Mgc_file"], @name);
```

2. This example filters out all objects in the current Navigator window that have either an *.attr* or a *.bak* extension. The objects are displayed in the Navigator window sorted first by type and then by name.

```
$setup_filter_active([".attr", ".bak"], @exclude, [],  
                    @type);
```

Related Topics

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

\$setup_filter_all()

Scope: dmgr_session_window

Specifies filters that determine which objects are displayed in all navigators that you open after running this function.

Usage

```
$setup_filter_all([name_wildcard], include_wildcard, [type_names], sort_by)
```

```
SETUp Filter All [name_wildcard] include_wildcard [type_names] sort_by
```

Arguments

- **name_wildcard**

A vector of strings that specifies which objects that are to be included or excluded from the navigator display. You can specify these strings using UNIX System V wildcards. If the `name_wildcard` field is empty, objects are filtered based only on their type. The format of this vector of strings is:

```
[“string1”, “string2”, ..., “stringN”]
```

String is a string that specifies the leaf name of the object. The default is the last string(s) specified or the vector `[*]` if you have not executed this function in the current session.

- **include_wildcard**

A name that indicates whether object names that match one or more strings in the `name_wildcard` pattern are included or excluded from the list of objects being displayed. The default is the last option specified or `@include` if you have not executed this function in the current session. Choose one of the following:

- **@exclude** — Excludes all objects which pass the `name_wildcard` filter.
- **@include** — Includes all objects which pass the `name_wildcard` filter.

- **type_names**

A vector of strings that specifies the design objects that are excluded based on their type. This argument defines the universe of design object types known to this function. If the `type_names` field is empty, all design object types are included and objects are filtered based only on their name. The format of this vector of strings is —

```
[“string1”, “string2”, ..., “stringN”]
```

String is a string that specifies the type of the object. The default is the last type(s) specified or the vector `[*]` if you have not executed this function in the current session.

- **sort_by**

A name that specifies whether objects are displayed in a list that is sorted first by type and then by name, or if they are displayed in alphabetical order with no regard for type. The default is the last option specified or `@name` if you have not executed this function in the current session.

Choose one of the following:

- **@name** — Objects are displayed in alphabetical order by design object name without regard for type.
- **@type** — All objects of a single type are displayed together in alphabetical order. The type groups are displayed alphabetically based on their type name.

Return Values

VOID.

Description

The interactive function `$setup_filter_all()` specifies filters, based on a design object's name and type, that determine which objects are displayed in all navigator windows you open after executing this function. The `type_names` argument defines the set of design objects that the `name_wildcard` and `include_wildcard` arguments operate on. That is, if you specify to include or exclude all design objects whose names match a specific pattern, this function only matches against those design objects which have successfully passed through the filters specified by the `type_names` argument. By default, no types are filtered out and navigator filters are set to include `"*"` which means that all files, except those that begin with a `"."`, are displayed.

This function sets filters on any navigators you open in the current session after executing this function. The filter settings are automatically saved in the Pyxis Project Manager default startup file *dmgr_default.startup*, so that you can set the navigator filters once and reapply the settings to each navigator in future sessions. If a navigator is in reference mode, the filters do not apply and the navigator display does not change.

If you have a custom startup file, your navigator filter settings in your default startup file are not used. If you have a custom startup file and you want your navigator filter settings to be used in subsequent sessions, copy your settings to your custom startup file.

If you want to change your navigator filter settings, you can either override a single navigator filter setting by executing the `$setup_filter_active()` function on the active Navigator window, or you can overwrite all navigator filter settings in your default startup file by executing the `$setup_filter_all()` function() again.

Examples

The following examples set filters on all navigators that are opened in the session following the execution of the function, and on any navigators that are open prior to the execution of the function that do not already have filter settings specified.

1. This example filters out all objects that are of some type other than either `"Dme_config_do"` and do not have a `"."` extension. The remaining objects are displayed alphabetically.

```
$setup_filter_all([".*"], @include, ["Dme_config_do"], @name);
```

2. This example filters out all objects that have either an `.attr` or a `.bak` extension. The remaining objects are displayed sorted first by type and then by name.

```
$setup_filter_all([".attr", ".bak"], @exclude, [], @type);
```

Related Topics

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

\$setup_default_editor()

Scope: dmgr_session_window

Prerequisite: You can use this function from any Pyxis Project Manager window.

Sets the value for the default editor in Pyxis Project Manager.

Usage

`$setup_default_editor(editortype, "editorpath", "readonlypath")`

SETUp DEfault Editor editortype editorpath readonlypath

Setup > Preferences > Session panel

Arguments

- **editortype**

A name that defines the type of editor that is invoked when you open an ASCII file. The default is @notepad. Choose one of the following:

- **@notepad** — The editor for the session is the Notepad editor. Default.
- **@user** — The editor for the session is user-defined.

- **editorpath**

A string that specifies the pathname of the user-defined ASCII editor. You need to supply this value only if you specified @user for the editortype argument. The pathname contained in this string must exist or an error is produced.

- **readonlypath**

A string that specifies the pathname of the read-only editor. You need to supply this value only if you specified @user for the editortype argument. The pathname contained in this string must exist or an error is produced.

Return Values

VOID.

Description

This function displays a dialog box that enables you to specify the default editor as the Notepad or as a user-specified editor. This function does error-checking to ensure that a usable user-specified editor is specified. Both the editorpath and readonlypath arguments must have valid pathnames to existing editors before the dialog box can be executed. If the pathname of either one of these arguments does not exist, an error message is produced and the execution of the **OK** button does not respond.

Examples

The following example specifies the default read/write editor as **vi** and the default read-only editor as **view**.

```
$setup_default_editor(@user, "/usr/bin/vi", "/usr/bin/view");
```

Related Topics

[\\$setup_iconic_window_layout\(\)](#)

[\\$setup_session_defaults\(\)](#)

[\\$setup_monitor\(\)](#)

[\\$setup_startup_windows\(\)](#)

\$setup_iconic_window_layout()

Scope: dmgr_session_window

Prerequisite: You can use this function from any Pyxis Project Manager window.

Specifies the layout features in iconic windows.

Usage

`$setup_iconic_window_layout(ystagger, xspacing, yspacing)`

SETUp IConic Window Layout ystagger xspacing yspacing

Setup > Preferences > Classic - Iconic Navigator panel

Arguments

- **ystagger**
A Boolean that specifies whether or not to stagger icons vertically in an iconic Navigator, Tools window, or Trash window. If @true, then the icons that appear in iconic navigators are staggered vertically so that design object names are less likely to overlap.
- **xspacing**
An integer that specifies the number of pixels in the horizontal gap between icons in an iconic Navigator, Tools, or Trash window.
- **yspacing**
An integer that specifies the number of pixels in the vertical gap between icons in an iconic Navigator, Tools window, or Trash window.

Return Values

VOID.

Description

The interactive function `$setup_iconic_window_layout()` enables you to customize the layout of your iconic windows. This function brings up a dialog box that enables you to specifying the distance between icons and to specify that icons are staggered vertically.

Examples

1. The following example sets up the iconic window layout to stagger icons and separate icons by 75 pixels on the horizontal and 75 pixels on the vertical.

`$setup_iconic_window_layout(@true, 75, 75);`
2. The following example sets up the iconic window layout to stagger icons and separate icons by 75 pixels on the horizontal and 75 pixels on the vertical, using command syntax from the popup command line.

setu ic w l true 75 75

Related Topics

[\\$setup_default_editor\(\)](#)

[\\$setup_monitor\(\)](#)

[\\$setup_session_defaults\(\)](#)

[\\$setup_startup_windows\(\)](#)

\$setup_invoke_tool()

Scope: dm_tool_tk

Prerequisite: This function is only available from within a qualification or termination script.

Sets the active tool viewpoint based on the tool viewpoint pathname and type specifications.

Usage

```
$setup_invoke_tool(“toolpath”, “tooltype”)
```

Arguments

- *toolpath*

A string that specifies the pathname of the active tool viewpoint. The default is “”, which is an empty string.

- *tooltype*

A string that specifies the type of the active tool viewpoint.

Return Values

VOID.

Examples

1. This example initializes the current tool viewpoint to “NULL”.

```
$setup_invoke_tool("");
```

2. This example searches the toolbox and sets the active tool viewpoint to the tool specified by the toolpath argument. Only the leaf name of the tool is specified; the tool type need not be provided.

```
$setup_invoke_tool("Editor");
```

3. This example sets the active tool viewpoint to the tool specified by the toolpath and tooltype arguments.

```
$setup_invoke_tool("$MGC_HOME/toolbox/Editor",  
"Mgc_default_ascii_editor");
```

\$\$setup_monitor()

Scope: dm_config_tk

Specifies the setup of configuration monitoring facilities.

Usage

```
$$setup_monitor(filter_errors, filter_warnings, filter_info, show_window, output_mode,
    verbosity)
```

Arguments

- **filter_errors**

A name that specifies whether errors and failures are displayed in the status reports produced by configuration operations. Choose one of the following:

- **@nofilter** — Includes error and failure information in configuration status reports. Default.
- **@filter** — Error and failure information is not included in configuration status reports.

- **filter_warnings**

A name that specifies whether warnings are displayed in the status reports produced by configuration operations. Choose one of the following:

- **@nofilter** — Includes warning information in configuration status reports. Default.
- **@filter** — Excludes warning information from configuration status reports.

- **filter_info**

A name that specifies whether informational messages are displayed in the status reports produced by configuration operations. Choose one of the following:

- **@nofilter** — Includes informational messages in configuration status reports. Default.
- **@filter** — Excludes informational messages from configuration status reports.

- **show_window**

A name that specifies whether the configuration monitor window is displayed automatically during configuration operations. Choose one of the following:

- **@noshow** — The configuration monitor window is not displayed during configuration operations. Default.
- **@show** — Displays the configuration monitor window automatically during configuration operations.

- **output_mode**

A name that specifies whether configuration monitoring status is output to the monitor window or to the transcript window. Choose one of the following:

- **@transcript** — Displays configuration status monitoring information in the transcript window.
- **@window** — Displays configuration status monitoring information in the configuration monitor window. Default.

- **verbosity**

An integer between 1 and 5 that specifies the amount of status information reported during configuration operations. If you specify this argument as 1, status reporting is minimal; only a summary of the number of errors, warnings, and failures is reported. If you specify this argument as 5, status reporting is verbose; all errors, warnings, and failures are reported, as well as additional information about the progress of the configuration operation. By default, the level of verbosity is 2.

Return Values

VOID.

Description

The toolkit function `$$setup_monitor()` enables you to customize your configuration monitoring facilities. Using this function, you can specify the types of status information that are reported about a configuration operation, the visibility of configuration monitoring output, the location of configuration monitoring output, and the verbosity of configuration monitoring output.

This function enables you to specify that classes of status information are excluded from the monitor output by setting filters on failures and errors, warnings, and informational messages. Although you specify to filter out specific classes of status information from the monitor output, the total count of failures, errors, and warnings is still reported; only the status messages themselves are not reported.

You can specify that the active configuration's monitor window is automatically displayed during a configuration operation, or you can specify that it is hidden until you explicitly display it using the **Windows > Open Session Monitor** menu item. You can also specify that configuration status information is written to the transcript area, instead of to the monitor window.

This function does not automatically save the settings you specify to your default startup file. If you want to save your specified settings, you must specify to save the settings to your default startup file.

Examples

This example sets the configuration monitor to display all errors, warnings, and informational messages in the configuration monitor window during configuration operations. Configuration monitoring is specified at the maximum level.

```
$$setup_monitor(@nofilter, @nofilter, @nofilter,@show,@window,5);
```

Related Topics

[\\$\\$set_monitor_flag\(\)](#)

[\\$setup_monitor\(\)](#)

[\\$setup_default_editor\(\)](#)

\$setup_monitor()

Scope: dmgr_session_window

Prerequisite: You can use this function from any Pyxis Project Manager window.

Specifies the setup of the configuration monitoring facilities.

Usage

```
$setup_monitor(filter_errors, filter_warnings, filter_info_msgs, show_status_area,  
               status_display, verbosity)
```

```
SETUp MOnitor filter_errors filter_warnings filter_info_msgs show_status_area status_display  
verbosity
```

Setup > Preferences > Monitor panel

Arguments

- **filter_errors**

A name that specifies whether errors are displayed in the configuration status reports produced by Configuration window operations. Choose one of the following:

- **@nofilter** — Includes error information in the configuration status reports. Default.
- **@filter** — Excludes errors information from the configuration status report.

- **filter_warnings**

A name that specifies whether warnings are displayed in the configuration status reports produced by configuration operations. Choose one of the following:

- **@nofilter** — Includes warning information in the configuration status reports. Default.
- **@filter** — Excludes warning information from the configuration status report.

- **filter_info_msgs**

A name that specifies whether informational messages are displayed in the configuration status reports produced by Configuration window operations. Choose one of the following:

- **@nofilter** — Includes informational messages in the configuration status reports. Default.
- **@filter** — Excludes informational messages from the configuration status report.

- **show_status_area**

A name that specifies whether the configuration monitor window is displayed automatically during Configuration window operations. Choose one of the following:

- **@noshow** — The configuration monitor window is not displayed during configuration operations. Default.

- **@show** — The configuration monitor window is automatically displayed during configuration operations.
- **status_display**

A name that specifies whether configuration status is output to the monitor window or to the transcript window. Choose one of the following:

 - **@transcript** — Displays configuration status information in the transcript window.
 - **@window** — Displays configuration status information in the configuration monitor window. Default.
- **verbosity**

An integer between 1 and 5 that specifies the amount of status information reported during configuration operations. If you specify this argument as 1, status reporting is minimal; only a summary of the number of errors, warnings, and failures is reported. If you specify this argument as 5, status reporting is verbose; all errors, warnings, and failures are reported, as well as additional information about the progress of the configuration operation. By default, the level of verbosity is 2.

Return Values

VOID.

Description

Using this function, you can specify the types of status information that are reported about a configuration operation, the visibility of configuration monitoring output, the location of configuration monitoring output, and the verbosity of configuration monitoring output.

This function enables you to specify that classes of status information are excluded from the monitor output by setting filters on failures and errors, warnings, and informational messages. Although you specify to filter out specific classes of status information from the monitor output, the total count of failures, errors, and warnings is still reported; only the status messages themselves are not reported.

You can specify that the active configuration's monitor window is automatically displayed during a configuration operation, or you can specify that it hidden until you explicitly display it using the **Windows > Open Session Monitor** menu item. You can also specify that configuration status information is written to the transcript area, instead of to the monitor window.

Examples

1. This example specifies a minimal level of configuration monitoring by filtering out all errors, warnings, and informational messages, specifying monitoring information to be output to the transcript window only, and setting monitoring verbosity to 1.

```
$setup_monitor(@filter, @filter, @filter, @noshow, @transcript, 1);
```

2. This example also specifies a minimal level of configuration monitoring by using command syntax from the popup command line.

setu mo -filter -filter -filter -noshow -transcript 1

Related Topics

[\\$\\$set_monitor_flag\(\)](#)

[\\$setup_session_defaults\(\)](#)

[\\$\\$setup_monitor\(\)](#)

\$setup_session_defaults()

Scope: dmgr_session_window

Prerequisite: You can use this function from any Pyxis Project Manager window.

Specifies the session defaults for the Pyxis Project Manager graphical user interface.

Usage

`$setup_session_defaults(click_speed, "input_device", layout, [menu_bar_visible],
 [window_title_visible], [message_area_visible], [softkey_area_visible], [palette_visible])`

SETUp SSession Defaults click_speed "input_device" layout [menu_bar_visible]
 [window_title_visible] [message_area_visible] [softkey_area_visible] [palette_visible]

Setup > Preferences > General panel and **Setup > Preferences > Dockables** panel

Arguments

- **click_speed**
 An integer that specifies the required double-click speed of the Select mouse button. The default is 125.
- **input_device**
 A string that specifies the input device to be used during your Pyxis Project Manager session. The default is either "MOUSE" or "X11_pointer", depending on your workstation.
- **layout**
 A name that specifies window layout in the Pyxis Project Manager session window.
 Choose one of the following:
 - **@stack** — Layers multiple windows on top of each other, with the most recently opened window on top.
 - **@left_right** — Places windows in the left half or the right half of the session window area. Default.
 - **@up_down** — Places windows in either the top half or the bottom half of the session window area.
 - **@quad** — Places windows in one of the four quadrants of the session window area.
 - **@prompt** — Enables you to specify a particular location for the window, using the Select mouse button to frame the desired window placement location.
- **menu_bar_visible**
 A vector that specifies whether the session menu bar is visible. By default, the menu bar is visible.

- **window_title_visible**
A vector that specifies whether the session title area is visible. By default, the window title is visible.
- **message_area_visible**
A vector that specifies whether the session message area is visible. By default, the message area is visible.
- **softkey_area_visible**
A vector that specifies whether the softkey area is visible. By default, the softkey area is not visible.
- **palette_visible**
A vector that specifies whether the palette area is visible. By default, the palette area is visible.

Return Values

VOID.

Description

The interactive function `$setup_session_defaults()` enables you to customize various aspects of the Pyxis Project Manager graphical interface. This function brings up a dialog box that enables you to set session defaults.

You may want to use this function to create Pyxis Project Manager startup scripts. A good way to determine the value of the arguments for this function is to find the settings that you want is by selecting the **Setup > Preferences > Session** panel menu item and transcribing the execution of this command.

When you run this function using the **Setup > Preferences > Session** panel menu item, you have the option of saving these values to your Pyxis Project Manager default startup file. If you run this function using the **MGC > Setup > Session** panel menu item, you cannot save the setup values you specify.

Examples

The following example specifies the mouse as the input device with a double-click speed of 100, quadrant window tiling, visible menu bars, an invisible session window title, and a visible message area.

```
$setup_session_defaults(100, "mouse", @quad, [@true], [@false], [@true], [@true]);
```

Related Topics

[\\$setup_default_editor\(\)](#)

[\\$setup_startup_windows\(\)](#)

[\\$setup_iconic_window_layout\(\)](#)

\$setup_startup_windows()

Scope: dmgr_session_window

Prerequisite: You can use this function from any Pyxis Project Manager window.

Specifies which windows are opened at invocation.

Usage

`$setup_startup_windows(open_tool_win, open_nav_win)`

SETUp STartup Windows open_tool_win open_nav_win

Setup > Preferences > Session panel

MGC > Setup > Session panel

Description

The interactive function `$setup_startup_windows()` enables you to specify whether navigators or a Tools window are opened when the Pyxis Project Manager application invokes. You have the option to specify whether a Tools window is opened or not opened at invocation, and whether an iconic or list navigator is opened or not opened at invocation.

Based on the options you specify, the `$setup_startup_windows()` function adds or does not add the `$open_navigator()` and `$open_tools_window()` functions to your default startup file *dmgr_default.startup*.

Arguments

- **open_tool_win**

A name that determines if the `$open_tools_window()` function is added to the default startup file. Choose one of the following:

- **@no** — Does not add the `$open_tools_window()` function to the default startup file.
- **@yes** — Adds the `$open_tools_window()` function to the default startup file.
Default.

- **open_nav_win**

A name that determines if the `$open_navigator()` function is added to the default startup file. The default is the previous value used for this argument. Choose one of the following:

- **@list** — Adds the `$open_navigator(@list)` function to the default startup file.
- **@iconic** — Adds the `$open_navigator(@iconic)` function to the default startup file.
Default.
- **@none** — Does not add the `$open_navigator()` function to the default startup file.

Return Values

VOID.

Examples

1. The following example specifies that a Tools window is not opened and a list navigator is opened during Pyxis Project Manager application invocation.

```
$setup_startup_windows(@no, @list);
```

2. The following example specifies that a Tools window is not opened and a list navigator is opened during Pyxis Project Manager application invocation, using command syntax from the popup command line.

```
setu st w no list
```

Related Topics

[\\$setup_iconic_window_layout\(\)](#)

[\\$setup_session_defaults\(\)](#)

\$show_all_files()

Scope: dm_project_area

Causes all non-directory file types to be shown within the Project Navigator window of the Pyxis Project Manager application.

Usage

`$show_all_files(show)`

Project Navigator popup menu > **Filter Views** > **Show All Files**

Arguments

- **show**

A Boolean that specifies whether all non-directory file types should be displayed or not.

Options include:

- **@true** — All files are shown and all other view type filter settings are ignored.
- **@false** — The files are shown as specified by the other view filter settings. Default,

Return Values

VOID.

Examples

This function can be called from a user startup file after first making the Project Navigator window as the active one:

```
$set_active_window("project_navigator");
```

```
$show_all_files(@true);
```

Related Topics

[\\$show_compiled_libs\(\)](#)

[\\$show_custom_views\(\)](#)

[\\$show_directories\(\)](#)

[\\$show_ext_libs\(\)](#)

[\\$show_language_views\(\)](#)

[\\$show_layout_views\(\)](#)

[\\$show_logic_views\(\)](#)

[\\$show_tech_libs\(\)](#)

\$show_compiled_libs()

Scope: dm_project_area

Sets the compiled library filter value within the Project Navigator window of the Pyxis Project Manager application.

Usage

`$show_compiled_libs(show)`

Project Navigator popup menu > **Filter Hierarchy** > **Show Compiled Libraries**

Arguments

- **show**

A Boolean that specifies whether compiled library directories should be displayed or not.

Options include:

- **@true** — Compiled libraries are displayed where they exist within open hierarchies.
- **@false** — Compiled libraries are not displayed. Default.

Return Values

VOID.

Examples

This function can be called from a user startup file after first making the Project Navigator window as the active one:

```
$set_active_window("project_navigator");
```

```
$show_compiled_libs(@true);
```

Related Topics

[\\$show_all_files\(\)](#)

[\\$show_custom_views\(\)](#)

[\\$show_directories\(\)](#)

[\\$show_ext_libs\(\)](#)

[\\$show_language_views\(\)](#)

[\\$show_layout_views\(\)](#)

[\\$show_logic_views\(\)](#)

[\\$show_tech_libs\(\)](#)

\$show_component_hierarchy()

Scope: dme_base_tk

Displays the design hierarchy of a design object in a component or IC design hierarchy window.

Usage

`$show_component_hierarchy("component_name", level, [filters], name_display)`

Component window popup menu > **Hide/Show Contents**

Component Hierarchy window popup menu > **Hide/Show Hierarchy**

Component Hierarchy window popup menu > **Show Levels**

Arguments

- **component_name**

A string that specifies the absolute pathname of the design object whose design hierarchy is to be displayed. The design object must be of the "mgc_component", "Ic_cell_template", "Ic_lib_template", or "Ic_persist_process" types.

- **level**

An integer that specifies the level at which hierarchical traversal stops. The default is level 2 which displays the selected component design object and the level directly beneath the component.

- **filters**

A vector of strings that specifies patterns that are used to filter design objects from the hierarchy window. If any portion of a design object's pathname matches a string in this vector, the design object is excluded from the hierarchy window's display. Filters always apply to sub-levels of the hierarchy; that is, you can never filter out the root object. You can specify the strings in this vector using UNIX System V regular expressions.

- **name_display**

A name that specifies whether to display the hierarchy using each design object's absolute pathname or using only a leaf name. The default is @leaf which specifies to display only the leaf name. Options include:

- **@full (-Full)** — The design object's hierarchy is displayed using each design object's absolute pathname.
- **@leaf (-Leaf)** — The design object's hierarchy is displayed using each design object's leaf name only.

Return Values

A vector of string vectors that specifies the component hierarchy window name and its entries:

```
[“hierarchy_window_name”, [“entry1”, entry1_level, “entry1_pathname”,  
    “entry1_leafname”], [“entry2”, entry2_level, “entry2_pathname”, “entry2_leafname”],  
    ..., [“entryN”, entryN_level, “entryN_pathname”, “entryN_leafname”]]
```

The `hierarchy_window_name` is a string that specifies the window name of the hierarchy window within the calling session. Each sub-vector consists of an entry name, an `entry_level`, an `entry_pathname`, and an `entry_leafname`.

The `entry_name` is a string that specifies the name of a single entry in the hierarchy window. Each `entry_level` is an integer that specifies the level at which the corresponding `entry_name` resides in the hierarchy. Each `entry_pathname` is a string that represents the absolute pathname of the `entry_name`. Each `entry_leafname` is a string that designates the leaf name of the corresponding `entry_name`.

Description

When you run the `$show_component_hierarchy()` function on an object of the “`mgc_component`” type, the logical design hierarchy of the component is displayed in a component hierarchy window. When you execute the `$show_component_hierarchy()` function on an object of the “`Ic_cell_template`”, “`Ic_lib_template`”, or “`Ic_persist_process`” types, the physical design hierarchy of the IC design object is displayed in an IC design hierarchy window.

Both the component hierarchy window and the IC design hierarchy window are referred to as a hierarchy window and the design object whose hierarchy they display is referred to as a component.

A components' design hierarchy is displayed as an indented list of names. You can specify that this function displays absolute pathnames for each of the components' subcomponents or that it displays only a leaf name for each of the subcomponents.

By default, this function displays the first two levels of a component design object's hierarchy without filtering out any design objects. You can specify that hierarchical descent proceed to a particular level of the design object's hierarchy by specifying the `level` argument. When you specify the `level` argument as a particular value, only the component design objects which exist at and above the specified level are displayed in the hierarchy window. If you specify the `level` argument as the default level 2, only the component design object specified in the `component_pathname` argument and the component design objects it contains in the level immediately below it are displayed in the hierarchy window. If you specify the `level` argument as 0, the entire hierarchy for the specified component design object is displayed.

You can also specify filters to exclude particular components from the hierarchy window by using the `filters` argument. If any portion of a component's pathname matches a string in the `filters` vector argument, that component is excluded from the hierarchy window's display.

In the component hierarchy window, the name of the schematic or VHDL architecture that was traversed to find the component is displayed to the right of the component in parenthesis. The VHDL architecture is designated by “`vhdl`”, and schematic models are indicated by a “`schm`”. All of the components with the same architectures that exist at the same hierarchical level are grouped together when they are displayed.

Examples

1. This example displays 2 levels of the logical design hierarchy for the “analog” component of the “mgc_component” type using the absolute pathname for each of the subcomponents without specifying any filters.

```
$show_component_hierarchy("$PROJECT_X/card_reader/analog",  
2, [], @full);
```

This example displays the following design hierarchy:

```
$PROJECT_X/card_reader/analog (schm:schematic)
$PROJECT_X//card_reader/analog/phase_det (schm:schematic)
$DPM_RLSLIB/parts/generic/inxhier (primitive)
$DPM_RLSLIB/parts/generic/outxhier (primitive)
$PROJECT_X/card_reader/analog/rf_amp (schm:schematic)
$PROJECT_X/card_reader/analog/demod (schm:schematic)
```

2. This example performs the same function as the previous example but specifies to traversal all levels.

```
$show_component_hierarchy("$PROJECT_X/card_reader/analog", 0, [], @full);
```

The preceding example displays the following design hierarchy:

```
$PROJECT_X/card_reader/analog (schm:schematic)
$PROJECT_X/card_reader/analog/phase_det (schm:schematic)
  $DPM_RLSLIB/parts/resistors/res.appl (primitive)
  $DPM_RLSLIB/parts/generic/vcc (primitive)
  $DPM_RLSLIB/parts/generic/inxhier (primitive)
  $DPM_RLSLIB/parts/generic/outxhier (primitive)
  $DPM_RLSLIB/parts/generic/gnd (primitive)
  $DPM_RLSLIB/parts/microcircuits/opamp (primitive)
$DPM_RLSLIB/parts/generic/inxhier (primitive)
$DPM_RLSLIB/parts/generic/outxhier (primitive)
$PROJECT_X/card_reader/analog/rf_amp (schm:schematic)
  $DPM_RLSLIB/parts/resistors/res.appl (primitive)
  $DPM_RLSLIB/parts/generic/gnd (primitive)
  $DPM_RLSLIB/parts/generic/outxhier (primitive)
  $DPM_RLSLIB/parts/capacitors/cap.appl (primitive)
  $DPM_RLSLIB/parts/capacitors/cap (primitive)
  $DPM_RLSLIB/parts/generic/inxhier (primitive)
  $DPM_RLSLIB/parts/generic/vcc (primitive)
  $DPM_RLSLIB/parts/transistors/pnp (primitive)
  $DPM_RLSLIB/parts/transistors/nfet (primitive)
$PROJECT_X/card_reader/analog/demod (schm:schematic)
  $DPM_RLSLIB/parts/transistors/npn (primitive)
  $DPM_RLSLIB/parts/generic/gnd (primitive)
  $DPM_RLSLIB/parts/resistors/res.appl (primitive)
  $DPM_RLSLIB/parts/capacitors/cap.appl (primitive)
  $DPM_RLSLIB/parts/generic/inxhier (primitive)
  $DPM_RLSLIB/parts/generic/vcc (primitive)
  $DPM_RLSLIB/parts/generic/outxhier (primitive)
  $DPM_RLSLIB/parts/diodes/diode (primitive)
```

3. This example displays 2 levels of the logical design hierarchy for the “cpu” component of the “mgc_component” type using the leaf name of each of the sub-components and without specifying any filters.

```
$show_component_hierarchy("$SANDBOX/cpu", 2, [], @leaf);
```

The preceding example returns the following vector in the transcript window:

```
//  [
//    "Text Report#4",
//    ["cpu", 0, "$SANDBOX/cpu", "cpu"],
//    ["..adder", 1, "$SANDBOX/adder", "adder"],
//    ["..portin", 1, "$GEN_LIB/portin", "portin"],
//    ["..portout", 1, "$GEN_LIB/portout", "portout"]
//  ]
```

4. This example displays all levels of the logical design hierarchy for the “analog” component of the “mgc_component” type. However, any sub-components whose pathnames contain the strings “in”, “out”, “vcc” or “gnd” are not displayed in the component hierarchy window.

```
$show_component_hierarchy("$PROJ_X/card_reader/analog",
0, ["in", "out", "vcc", "gnd"], @full);
```

5. This example displays all levels of the physical design hierarchy for the “4_bit_adder” component of the “Ic_cell_template” type.

```
$show_component_hierarchy("$PROJ_X/layout/4_bit_adder",
0, [], @leaf);
```

This example displays the following design hierarchy:

```
4_bit_adder
  process
    a9500
    a9510
  sandia_lib
    a1120
      pcap
      ctbot
      ctrl
        res7
      orcap
    accumulator
      a1310
      a9450
    cla
      a2311
    add32
      cla
        a2311
    control
      placore
        ll
        ctrl
          res7
        orcap
```

Related Topics

[`\$descend_hierarchy_one_level\(\)`](#)

[`\$descend_hierarchy_specify_level\(\)`](#)

\$show_custom_views()

Scope: dm_project_area

Setup > Preferences > Project Navigator panel

Sets the custom views filter within the Project Navigator window of the Pyxis Project Manager application.

Usage

`$show_custom_views(show)`

Project Navigator popup menu > **Filter Views > Show Custom Views**

Arguments

- **show**

A Boolean that specifies whether or not custom view types should be displayed. Options include:

- **@true** — Custom views are shown. Default.
- **@false** — Custom views are not shown.

Return Values

VOID.

Description

Custom views include any view types defined using the Pyxis Project Manager application's type registration facilities or any other types that contain the type property “custom_view_type”.

Examples

This function can be called from a user startup file after first making the Project Navigator window as the active one:

```
$set_active_window("project_navigator");
```

```
$show_custom_views(@true);
```

Related Topics

[\\$show_all_files\(\)](#)

[\\$show_compiled_libs\(\)](#)

[\\$show_directories\(\)](#)

[\\$show_ext_libs\(\)](#)

[\\$show_language_views\(\)](#)

[\\$show_layout_views\(\)](#)

[\\$show_logic_views\(\)](#)

[\\$show_tech_libs\(\)](#)

\$show_directories()

Scope: dm_project_area

Sets the untyped directory filter within the Project Navigator window of the Pyxis Project Manager application.

Usage

`$show_directories(show)`

Project Navigator popup menu > **Filter Hierarchy** > **Show Directories**

Arguments

- **show**

A Boolean that specifies whether or not untyped directories should be displayed. Options include:

- **@true** — Compiled directories are shown.
- **@false** — Compiled directories are not shown. Default.

Return Values

VOID.

Description

Untyped directories are not shown by default within the Project Navigator but it may be desirable to see them at times, such as when viewing or editing NCF files.

Examples

This function can be called from a user startup file after first making the Project Navigator window as the active one:

```
$set_active_window("project_navigator");
```

```
$show_directories(@true);
```

Related Topics

[\\$show_all_files\(\)](#)

[\\$show_compiled_libs\(\)](#)

[\\$show_custom_views\(\)](#)

[\\$show_ext_libs\(\)](#)

[\\$show_language_views\(\)](#)

[\\$show_layout_views\(\)](#)

[\\$show_logic_views\(\)](#)

[\\$show_tech_libs\(\)](#)

\$show_ext_libs()

Scope: dm_project_area

Sets the external library filter within the Project Navigator window of the Pyxis Project Manager application.

Usage

`$show_ext_libs(show)`

Project Navigator popup menu > **Filter Hierarchy** > **Show External Libraries**

Arguments

- **show**

A Boolean that specifies whether or not external libraries should be displayed.

- **@true** — External libraries are shown within projects and other external libraries that include them. Default.
- **@false** — Only external libraries that are open within their own hierarchy are shown.

Return Values

VOID.

Examples

This function can be called from a user startup file after first making the Project Navigator window as the active one:

```
$set_active_window("project_navigator");
```

```
$show_ext_libs(@true);
```

Related Topics

[\\$show_all_files\(\)](#)

[\\$show_compiled_libs\(\)](#)

[\\$show_custom_views\(\)](#)

[\\$show_directories\(\)](#)

[\\$show_language_views\(\)](#)

[\\$show_layout_views\(\)](#)

[\\$show_logic_views\(\)](#)

[\\$show_tech_libs\(\)](#)

\$show_invalid_language_views()

Scope: dmgr_project_model

Highlights invalid language views and symbols for the specified objects in a set of HDL files.

Usage

```
$show_invalid_language_views([invalid_obj_paths_types_and_error_msg])
```

Arguments

- **invalid_obj_paths_types_and_error_msg**

A vector of strings specifying objects in the current hierarchy in the format:

```
[[“obj1_path”, “obj1_type”, “obj1_error_msg”], [“obj2_path”, “obj2_type”,  
“obj2_error_msg”],...].
```

Description

Given a set of HDL files, this function highlights invalid language views and symbols for the specified objects. If the error message (error_msg) is non-empty, it highlights the given object with red color and sets the specified error_msg as its tooltip. Otherwise, it highlights the specified object in black.

Examples

The following example highlights the invalid “delay_block” view of the “mgc_verilog” type in red color all through the hierarchy until the project root, and sets “Need to be Compiled” as its tooltip:

```
$show_invalid_language_views(["/test_flow/flow_lib/delay_block/delay_block",  
"mgc_verilog", "Need to be Compiled."]);
```

Related Topics

[\\$check_language_views\(\)](#)

[\\$check_language_view\(\)](#)

[\\$generate_language_view\(\)](#)

\$show_language_views()

Scope: dm_project_area

Sets the language views filter within the Project Navigator window of the Pyxis Project Manager application.

Usage

`$show_language_views(show)`

Project Navigator popup menu > **Filter Views** > **Show Language Views**

Arguments

- **show**

A Boolean that specifies whether or not language view files should be displayed. Options include:

- **@true** — Language views are shown. Default.
- **@false** — Language views are not shown.

Description

Language views include Verilog, Verilog-A, Verilog AMS, VHDL, VHDL-AMS, and SPICE.

Return Values

VOID.

Examples

This function can be called from a user startup file after first making the Project Navigator window as the active one:

```
$set_active_window("project_navigator");
```

```
$show_language_views(@true);
```

Related Topics

[\\$show_all_files\(\)](#)

[\\$show_compiled_libs\(\)](#)

[\\$show_custom_views\(\)](#)

[\\$show_directories\(\)](#)

[\\$show_ext_libs\(\)](#)

[\\$show_layout_views\(\)](#)

[\\$show_logic_views\(\)](#)

[\\$show_tech_libs\(\)](#)

\$show_layout_views()

Scope: dm_project_area

Sets the layout view filter within the Project Navigator window of the Pyxis Project Manager application.

Usage

```
$show_layout_views(show)
```

Project Navigator popup menu > **Filter Views** > **Show Layout Views**

Arguments

- **show**

A Boolean that specifies whether or not physical design objects should be displayed.

Options include:

- **@true** — Layout views are shown. Default.
- **@false** — Layout views are not shown.

Return Values

VOID.

Examples

This function can be called from a user startup file after first making the Project Navigator window as the active one:

```
$set_active_window("project_navigator");
```

```
$show_layout_views(@true);
```

Related Topics

[\\$show_all_files\(\)](#)

[\\$show_compiled_libs\(\)](#)

[\\$show_custom_views\(\)](#)

[\\$show_directories\(\)](#)

[\\$show_ext_libs\(\)](#)

[\\$show_language_views\(\)](#)

[\\$show_logic_views\(\)](#)

[\\$show_tech_libs\(\)](#)

\$\$show_location_map()

Scope: dme_dn_tk

Displays, in a read-only window, the absolute path of the location map in use and the value of the current location map entries.

Usage

`$$show_location_map()`

Arguments

None.

Return Values

The string “location_map”.

Examples

This example opens a read-only window containing the current location map entries in memory.

```
$$show_location_map();
```

Related Topics

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

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

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

[\\$show_location_map\(\)](#)

\$show_location_map()

Scope: session_area

Prerequisite: You can use this function from any Pyxis Project Manager window.

Displays, in a read-only window, the absolute pathname of the current location map in use and the value of its entries in location map format.

Usage

`$show_location_map()`

SHOW LLocation Map

MGC > Location Map > Show Location Map

Arguments

None.

Return Values

VOID.

Description

Use the file menu to save your location map display to a file for future use. This is especially useful if you have experimented with `$change_location_map_entry()` and want to save the state of the location map.

Examples

This example opens a read-only window containing the current location map entries in memory.

```
$show_location_map();
```

Related Topics

[\\$change_location_map_entry\(\)](#)

[\\$\\$show_location_map\(\)](#)

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

[\\$export_location_map\(\)](#)

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

\$show_logic_views()

Scope: dm_project_area

Sets the logic views filter within the Project Navigator window of the Pyxis Project Manager application.

Usage

`$show_logic_views(show)`

Project Navigator popup menu > **Filter Views** > **Show Logic Views**

Arguments

- **show**

A Boolean that specifies whether or not logical design objects should be displayed. Options include:

- **@true** — Logic views are shown. Default.
- **@false** — Logic views are not shown.

Return Values

VOID.

Description

Logic view types include schematics, symbols, and design viewpoints.

Examples

This function can be called from a user startup file after first making the Project Navigator window as the active one:

```
$set_active_window("project_navigator");  
$show_logic_views(@true);
```

Related Topics

[\\$show_all_files\(\)](#)

[\\$show_compiled_libs\(\)](#)

[\\$show_custom_views\(\)](#)

[\\$show_directories\(\)](#)

[\\$show_ext_libs\(\)](#)

[\\$show_language_views\(\)](#)

[\\$show_layout_views\(\)](#)

[\\$show_tech_libs\(\)](#)

\$show_references()

Scope: dmgr_model

Prerequisite: To use this function, you must be in an active Navigator window and must select at least one design object.

Displays the references of the selected design object.

Usage

`$show_references()`

SHOW REferences

Report > Show References

Navigator window popup menu > **Report > Show References**

Arguments

None.

Return Values

A string that specifies the name of the Reference window. For example, if no other Reference windows are open, this function returns the string “reference”. If a Reference window is currently open, this function returns “reference#2”.

Description

The Reference window contains the absolute pathname and version number of each referenced design object. If a reference points to the current version of a design object, a “[]” follows the pathname of the object.

If you execute this function on a design object that does not have references, an empty Reference window is shown.

You can press the ShowReferences key, as a shortcut method for executing the `$show_references()` function, as shown in “[Key Names Mapped to Functions & Scopes](#)” on page 804.

Examples

1. This example displays the references of the selected design object.

`$show_references();`

2. This example displays the references of the selected design object by using command syntax from the popup command line.

`sho re`

Related Topics

[\\$add_reference\(\)](#)

[\\$delete_reference\(\)](#)

[\\$report_reference_info\(\)](#)

\$show_monitor()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window.

Makes the configuration monitor window visible.

Usage

`$show_monitor()`

SHOw MOonitor

Arguments

None.

Return Values

VOID.

Description

This function removes the active configuration's window from view and displays the active configuration's monitor window in its place. The configuration's monitor window is automatically cleared upon each new configuration operation.

You can customize your session setup values to specify that the configuration monitor window is either visible or hidden during configuration operations.

You can use the Monitor button as a shortcut method for executing the `$show_monitor()` function, as shown in “[Key Names Mapped to Functions & Scopes](#)” on page 804.

Examples

1. This example displays the active configuration's monitor window.

```
$show_monitor();
```

2. This example displays the active configuration's monitor window, by using command syntax from the popup command line.

```
sho mo
```

Related Topics

[\\$\\$clear_monitor\(\)](#)

[\\$open_session_monitor\(\)](#)

[\\$hide_monitor\(\)](#)

[\\$\\$writeln_monitor\(\)](#)

[Change Your Session Setup](#)

\$show_tech_libs()

Scope: dm_project_area

Sets the technology library filter within the Project Navigator window of the Pyxis Project Manager application.

Usage

`$show_tech_libs(show)`

Project Navigator popup menu > **Filter Hierarchy** > **Show Tech Libraries**

Arguments

- **show**

A Boolean that specifies whether or not technology libraries should be displayed. Options include:

- **@true** — The technology libraries referenced by all opened projects and external libraries are displayed. Default.
- **@false** — Only technology libraries that are open within their own hierarchies are displayed.

Return Values

VOID.

Examples

This function can be called from a user startup file after first making the Project Navigator window as the active one:

```
$set_active_window("project_navigator");
```

```
$show_tech_libs(@true);
```

Related Topics

[\\$show_all_files\(\)](#)

[\\$show_compiled_libs\(\)](#)

[\\$show_custom_views\(\)](#)

[\\$show_directories\(\)](#)

[\\$show_ext_libs\(\)](#)

[\\$show_language_views\(\)](#)

[\\$show_layout_views\(\)](#)

[\\$show_logic_views\(\)](#)

\$show_versions()

Scope: dmgr_model

Prerequisite: To use this function, you must be in an active Navigator window and must select at least one design object.

Displays all versions of the selected design object.

Usage

\$show_versions()

SHOW VErSIONS

Report > Show Versions

Navigator window popup menu > **Report > Show Versions**

Arguments

None.

Return Values

A string that specifies the name of the Version window. For example, if no other Version windows are open, this function returns the string “version”. If a Version window is currently open, this function returns “version#2”.

Examples

1. This example displays the versions of the selected design object.

```
$show_versions();
```

2. This example displays the versions of the selected design object by using command syntax from the popup command line.

```
sho ve
```

Related Topics

[\\$delete_version\(\)](#)

[\\$unfreeze_version\(\)](#)

[\\$freeze_version\(\)](#)

\$trash_object()

Scope: dmgr_model

Prerequisite: To use this function, you must be in an active Navigator window and must select at least one design object.

Moves the selected design objects to the Trash window.

Usage

\$trash_object()

TRAsh OBject

Arguments

None.

Return Values

VOID.

Description

To remove the design objects from the Trash window, you use the \$suntrash_object() function. You can also remove the selected design object from the Trash by dragging its icon to another window.

Examples

1. This example places the selected design objects in the Trash window.

```
$trash_object();
```

2. This example places the selected design objects in the Trash window by using command syntax from the popup command line.

```
tra ob
```

Related Topics

[\\$empty_trash\(\)](#)

[\\$suntrash_object\(\)](#)

\$\$unfreeze_configuration()

Scope: dm_config_tk

Unfreezes all design object versions in the active configuration.

Usage

\$\$unfreeze_configuration()

Arguments

None.

Return Values

VOID.

Examples

This example opens the “config_beta” configuration and unfreezes it.

```
$$open_configuration("$PROJECT_XYZ/d1/config_beta", 0, @read_write);
```

```
$$unfreeze_configuration();
```

Related Topics

[**\\$\\$freeze_configuration\(\)**](#)

\$unfreeze_configuration()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window.

Unfreezes all design object versions in the active Configuration window.

Usage

\$unfreeze_configuration()

UNFREeze CONfiguration

File > Unfreeze

Configuration window popup menu > **Global Operations > Unfreeze**

Arguments

None.

Return Values

VOID.

Description

You can interrupt the execution of the \$unfreeze_configuration() function at any time, by pressing the Kill key. When you halt the unfreeze operation, a dialog box appears prompting you to abort or to continue the operation. If you choose to abort, the operation is halted. Design object versions which have been unfrozen prior to the interrupt remain unfrozen; all others remain frozen.

Examples

1. This example unfreezes the design object versions in the active Configuration window.

```
$unfreeze_configuration();
```

2. This example unfreezes design object versions in the active Configuration window by using command syntax from the popup command line.

```
unfre co
```

Related Topics

[\\$freeze_configuration\(\)](#)

[\\$report_configuration_info\(\)](#)

\$\$unfreeze_version()

Scope: dme_do_tk

Prerequisite: To use this function, you must be in an active version window.

Unfreezes a frozen version of the specified design object.

Usage

`$$unfreeze_version("obj_name", "obj_type", version)`

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **obj_type**
A string that specifies the type of the design object.
- **version**
An integer that specifies the version number of the design object. The default is the current version.

Return Values

VOID.

Description

The toolkit function `$$unfreeze_version()` unfreezes the specified frozen version of the specified design object. By default, this function unfreezes the current version of the specified design object. To unfreeze a previous version, you must specify the version argument. If the version is not frozen, this function has no effect.

The next time you use the `$$save_object()` function, any unfrozen versions beyond the default version depth are deleted without warning. For example, if you unfreeze a version that falls beyond the default version depth, that version is deleted by a subsequent `$$save_object()` operation.

Suppose you have a design object whose version depth is set to 2. However, a frozen version exists so that you have version 1, which is frozen; version 2; and version 3. If you unfreeze version 1, the next time you perform a save on the design object, version 1 is deleted, leaving you with versions 2 and 3.

Examples

This example unfreezes version 6 of the design object *library*.

```
$$unfreeze_version("$PROJECT_XYZ/d1/parts/library","Mgc_file",6);
```

Related Topics

[\\$\\$delete_version\(\)](#)

[\\$\\$freeze_version\(\)](#)

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

[\\$\\$save_object\(\)](#)

[\\$\\$get_version_depth\(\)](#)

[\\$\\$revert_version\(\)](#)

[\\$\\$set_version_depth\(\)](#)

\$unfreeze_version()

Scope: dmgr_version_model

Prerequisite: To use this function, you must be in an active version window.

Unfreezes a frozen version of the selected design object.

Usage

\$unfreeze_version()

UNFREeze VErSION

Edit > Unfreeze

Versions window popup menu > **Unfreeze**

Description

The interactive function \$unfreeze_version() unfreezes the frozen version of the selected design object. If, by unfreezing a version, you cause the design object to exceed its default version depth, any unfrozen objects beyond the default version depth are deleted the next time you perform a save.

Suppose you have a design object whose version depth is set to 2. However, a frozen version exists so that you have version 1, which is frozen; version 2; and version 3. If you unfreeze version 1, the next time you perform a save, version 1 is deleted, leaving you with versions 2 and 3.

Arguments

None.

Return Values

VOID.

Examples

1. This example unfreezes the selected frozen version.

```
$unfreeze_version();
```

2. This example unfreezes the selected version by using command syntax from the popup command line.

```
unfre ve
```

Related Topics

[\\$freeze_version\(\)](#)

\$\$unlock_configuration()

Scope: dm_config_tk

Unlocks all design object versions in the active configuration.

Usage

\$\$unlock_configuration()

Arguments

None.

Return Values

VOID.

Examples

This example opens the configuration *config_zoo* and unlocks the active configuration.

```
$$open_configuration("$PROJECT_XYZ/d1/config_zoo",0,@read_write);
```

```
$$unlock_configuration();
```

Related Topics

[\\$\\$lock_configuration\(\)](#)

\$unlock_configuration()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window.

Unlocks all design object versions in the active Configuration window.

Usage

\$unlock_configuration()

UNLOCK Configuration

File > Unlock Configuration

Configuration window popup menu > **Global Operations > Unlock Configuration**

Arguments

None.

Return Values

VOID.

Description

When you unlock a configuration, the design object versions that are part of the configuration, can be accessed by other users.

You can interrupt the execution of the \$unlock_configuration() function, at any time, by pressing the Kill key. When you interrupt the unlock operation, a dialog box appears prompting you whether to abort or to continue the operation. If you choose to abort, the operation is halted. Design object versions that have been unlocked prior to the interrupt remain unlocked; all others remain locked.

Examples

1. This example unlocks the contents of the active Configuration window.

\$unlock_configuration();

2. This example unlocks the contents of the active Configuration window by using command syntax from the popup command line.

unloc co

Related Topics

[\\$lock_configuration\(\)](#)

\$\$unlock_object()

Scope: dme_do_tk

Unlocks the specified design object.

Usage

```
$$unlock_object("obj_name", "type")
```

Arguments

- **obj_name**
A string that specifies the pathname of the design object.
- **type**
A string that specifies the type of the design object.

Return Values

A Boolean specifying whether the lock was released on the object.

Description

The toolkit function `$$unlock_object()` unlocks the specified design object, which was previously locked by the `$$lock_object()` function. The functions `$$save_object()`, `$$lock_object()`, and `$$unlock_object()` can be used when you perform multiple operations on a single design object. You can lock the design object once, perform all your operations, saving as needed, and then unlock the object when you are finished.

When you lock an object using the `$$lock_object()` function, you can use the `count_read_req` argument to specify that the number of requests you make for a read-only lock, on a single object, are counted. Subsequent executions of the `$$unlock_object()` function decrement the count. As shown in the examples, you can use the returned value to determine whether all locks have been removed.

Examples

1. This example unlocks the design object *file*.

```
$$unlock_object("$PRJ/file", "Mgc_file");
```

2. This example performs two unlocks on the design object "*\$PRJ/file*", which had two read-only locks added to it using the `$$lock_object()` function with the mode option set to `@no_modify` and the `count_read_req` option set to true.

```
$$lock_object("$PRJ/file","Mgc_file",@no_modify,@true);  
// count --> 1  
$$lock_object("$PRJ/file","Mgc_file",@no_modify,@true);  
// count --> 2  
$$unlock_object("$PRJ/file","Mgc_file");  
// count --> 1, ret = @false  
$$unlock_object("$PRJ/file","Mgc_file");  
// count --> 0, ret = @true
```

Related Topics

[\\$\\$lock_object\(\)](#)

[\\$\\$save_object\(\)](#)

\$unselect_all()

Scope: dm_config_window, dm_iconic_area, dm_list_area, or dmgr_type_area

Prerequisite: To use this function, you must be in an active Navigator, Reference, Tool, Trash, Version or Configuration window.

Unselects all selected design objects.

Usage

\$unselect_all()

UNSElect All

File > Unselect > All

Navigator window popup menu > **Unselect > All**

References window popup menu > **Unselect All**

Tools window popup menu > **Unselect All**

Versions window popup menu > **Unselect All**

Configuration window popup menu > **Unselect > All**

Arguments

None.

Return Values

VOID.

Description

If the active window does not contain at least one previously selected design object, the **Unselect All** popup menu item is grayed out.

You can press the UnselectAll key, as a shortcut method for executing the \$unselect_all() function, as shown in “[Key Names Mapped to Functions & Scopes](#)” on page 804.

Examples

1. This example unselects all selected design objects in the active window.

```
$unselect_all();
```

2. This example unselects all selected design objects in the active window by using command syntax from the popup command line.

```
unsel al
```

Related Topics

[`\$select_all\(\)`](#)

[`\$unselect_by_name\(\)`](#)

[`\$unselect_by_type\(\)`](#)

[`\$unselect_config_entry\(\)`](#)

[`\$unselect_reference\(\)`](#)

[`\$unselect_tool\(\)`](#)

[`\$unselect_trash_object\(\)`](#)

[`\$unselect_version\(\)`](#)

\$unselect_by_name()

Scope: dmgr_iconic_area, dmgr_list_area, or dm_config_window

Prerequisite: To use this function, you must be in either an active Navigator or an active Configuration window that has at least one previously selected object.

Unselects the specified design object based on its name.

Usage

Navigator window:

```
$unselect_by_name("name", "type")  
UNSElect BY Name name type
```

Configuration window:

```
$unselect_by_name("name", "type", version)  
UNSElect BY Name name type version
```


File > Unselect > By Name

Navigator window popup menu > **Unselect > By Name**

Configuration window popup menu > **Unselect > By Name**

Arguments

- **name**
A string that specifies the absolute pathname or the leaf name of the design object. You can specify the name by using UNIX System V wildcards.
- **type**
A string that specifies the type of the design object. You can specify the type by using UNIX System V wildcards.
- **version**
An integer that specifies the version number of the configuration entry.

 **Note** This argument is only valid in the Configuration window.

If the version is not specified, by default, the version is ignored and all entries that match the specified name and type are unselected.

Return Values

VOID.

Description

If you omit either the name or type argument, this function unselects all design objects that match the pattern of the argument specified.

In the Configuration window, if the version argument is not specified, by default, the version is ignored and all entries that match the name and type argument are selected. To unselect a specific version, you must specify the version argument.

Examples

1. This example unselects the “logo” design object of “Mgc_file” type.

```
$unselect_by_name("logo", "Mgc_file");
```

2. This example unselects the same design object by using command syntax from the popup command line.

```
unsel by n logo Mgc_file
```

3. This example uses a wildcard to unselect all design objects that begin with “config”.

```
unsel by n config*
```

Related Topics

[\\$select_all\(\)](#)

[\\$select_by_name\(\)](#)

[\\$select_config_entry\(\)](#)

[\\$select_reference\(\)](#)

[\\$select_tool\(\)](#)

[\\$select_trash_object\(\)](#)

[\\$select_version\(\)](#)

[\\$unselect_all\(\)](#)

[\\$unselect_by_name\(\)](#)

[\\$unselect_by_type\(\)](#)

[\\$unselect_tool\(\)](#)

[\\$unselect_trash_object\(\)](#)

\$unselect_by_type()

Scope: dmgr_iconic_area, dmgr_list_area, or dm_config_window

Prerequisite: To use this function, you must be in either an active Navigator or an active Configuration window that has at least one previously selected object.

Unselects the specified design objects based on their type.

Usage

`$unselect_by_type("type")`

UNSElect BY Type type

File > Unselect > By Type

Navigator window popup menu > **Unselect > By Type**

Configuration window popup menu > **Unselect > By Type**

Arguments

- **type**
A string that specifies the type of the design object. You can specify the type by using UNIX System V wildcards.

Return Values

VOID.

Examples

1. This example unselects all design objects of the "Mgc_file" type.

```
$unselect_by_type("Mgc_file");
```

2. This example selects the same design object by using command syntax from the popup command line.

```
unsel by t Mgc_file
```

3. This example uses a wildcard to unselect all design objects that have a type name that begins with the "Mgc" string.

```
unsel by t Mgc*
```

Related Topics

[\\$select_all\(\)](#)

[\\$select_by_name\(\)](#)

[\\$select_by_type\(\)](#)

[\\$select_trash_object\(\)](#)

[\\$select_version\(\)](#)

[\\$unselect_all\(\)](#)

[\\$select_config_entry\(\)](#)

[\\$unselect_by_name\(\)](#)

[\\$select_reference\(\)](#)

[\\$unselect_trash_object\(\)](#)

[\\$select_tool\(\)](#)

\$unselect_config_entry()

Scope: dm_config_window

Prerequisite: To use this function, you must be in a Configuration window that has at least one previously selected configuration entry.

Unselects all selected configuration entries.

Usage

`$unselect_config_entry("name", "type", version)`

UNSElect COnfig Entry name type *version*

Arguments

- **name**
A string that specifies the name of the configuration entry. You must specify the absolute pathname of the configuration entry. You can specify the name by using UNIX System V wildcards.
- **type**
A string that specifies the type of the configuration entry. You can specify the type by using UNIX System V wildcards.
- ***version***
An integer that specifies the version number of the configuration entry. If the version is not specified, by default, the version is ignored and all entries that match the specified name and type are unselected.

Return Values

VOID.

Description

If the version argument is not specified, by default, the version is ignored and all entries that match the name and type argument are unselected. To unselect a specific version, you must specify the version argument.

Examples

1. This example unselects version 2 of the configuration entry *mars*.

```
$unselect_config_entry("$PROJECT_XYZ/project/mars",  
    "Image_file", 2);
```

2. This example unselects the same configuration entry by using command syntax from the popup command line.

```
unsel co e $PROJECT_XYZ/project/mars Image_file 2
```

Related Topics

[`\$select_all\(\)`](#)

[`\$select_by_name\(\)`](#)

[`\$select_by_type\(\)`](#)

[`\$select_config_entry\(\)`](#)

[`\$select_tool\(\)`](#)

[`\$unselect_all\(\)`](#)

[`\$unselect_by_name\(\)`](#)

[`\$unselect_by_type\(\)`](#)

[`\$unselect_reference\(\)`](#)

[`\$unselect_tool\(\)`](#)

[`\$unselect_version\(\)`](#)

[Configuration Management Concepts](#)

\$unselect_object()

Scope: dmgr_iconic_area or dmgr_list_area

Prerequisite: To use this function, you must be in an active Navigator window.

Unselects the selected or specified design object.

Usage

`$unselect_object("name", "type")`

UNSElect OBject name *type*

File > Unselect

Arguments

- **name**
A string that specifies the name of the design object. You can specify the name by using UNIX System V wildcards.
- **type**
A string that specifies the type of the design object. You can specify the type by using UNIX System V wildcards. The default is the empty string "".

Return Values

VOID.

Description

If you omit either the name or the type argument, this function unselects all design objects that match the pattern of the name argument specified.

Examples

1. This example unselects the "mars" design object.

```
$unselect_object("mars", "Image_file");
```

2. This example unselects the same design object by using command syntax from the popup command line.

```
unsel ob mars Image_file
```

3. This example uses a wildcard to unselect all design objects that begin with "config".

```
unsel ob config*
```

Related Topics

[\\$select_by_name\(\)](#)

[\\$select_by_type\(\)](#)

[\\$select_config_entry\(\)](#)

[\\$select_tool\(\)](#)

[\\$unselect_all\(\)](#)

[\\$unselect_by_name\(\)](#)

[\\$unselect_by_type\(\)](#)

[\\$unselect_reference\(\)](#)

[\\$unselect_tool\(\)](#)

[\\$unselect_version\(\)](#)

\$unselect_reference()

Scope: dmgr_reference_list_area

Prerequisite: To use this function, you must be in an active Reference window with at least one previously selected reference.

Unselects the selected or specified reference.

Usage

`$unselect_reference("to_name", "to_type", to_version)`

`UNSElect REference to_name to_type to_version`

Arguments

- **to_name**
A string that specifies the absolute pathname of the target design object. You can specify the name by using UNIX System V wildcards.
- **to_type**
A string that specifies the type of the target design object. You can specify the type by using UNIX System V wildcards.
- **to_version**
An integer that specifies the version number of the target design object. The default is 0, which represents the current version.

Return Values

VOID.

Description

If you specify the `to_type` argument, this function unselects only references that point to design objects of that type. If you omit the `to_type` argument, this function unselects all references that point to design objects whose name matches the `to_name` argument.

By default, this function unselects the reference to the current version of the target object. To unselect a reference to a previous version of the target design object, you must specify the `to_version` argument.

Examples

1. This example unselects the reference to the current version of the “planets” design object.

```
$unselect_reference("$PROJ_XYZ/proj/example/planets",  
"Image_file", 0);
```

2. This example unselects the same reference by using command syntax from the popup command line.

`unsel re $PROJ_XYZ/proj/example/planets Image_file 0`

Related Topics

[\\$add_reference\(\)](#)

[\\$select_reference\(\)](#)

[\\$delete_reference\(\)](#)

[\\$show_references\(\)](#)

[References](#)

\$unselect_tool()

Scope: dmgr_tool_area

Prerequisite: To use this function, you must be in an active Tools window with at least one previously selected tool.

Unselects the selected or specified tool.

Usage

`$unselect_tool("name", "type")`

UNSElect TOol name *type*

Arguments

- **name**

A string that specifies the name of the tool. You can specify the name by using UNIX System V wildcards.

- **type**

A string that specifies the type of the tool. You can specify the type by using UNIX System V wildcards. The default is "", which is an empty string.

If you specify an empty string, this function unselects the first tool whose name matches the name you specified. If you use a wildcard to specify the type, this function unselects the first tool whose name matches the wildcard pattern you specified.

Return Values

VOID.

Examples

1. This example unselects the "dss" tool.

```
$unselect_tool("dss", "mgc_dss_tool");
```

2. This example unselects the same tool by using command syntax from the popup command line.

```
unsel too dss mgc_dss_tool
```

Related Topics

[\\$select_reference\(\)](#)

[\\$unselect_toolbox\(\)](#)

[\\$unselect_config_entry\(\)](#)

[\\$unselect_version\(\)](#)

\$unselect_toolbox()

Scope: dmgr_toolbox_area

Prerequisite: To use this function, you must be in an active Tools window that currently shows toolboxes.

Unselects the selected or specified toolbox.

Usage

`$unselect_toolbox("name")`

UNSElect TOOLBox name

Arguments

- **name**

A string that specifies the absolute pathname of the toolbox. You can specify the name by using UNIX System V wildcards.

Return Values

VOID.

Examples

1. This example unselects the *\$PROJ_XYZ/proj/toolbox1* toolbox.

```
$unselect_toolbox("$PROJ_XYZ/proj/toolbox1");
```

2. This example unselects the same toolbox by using command syntax from the popup command line.

```
unsel toolb $PROJ_XYZ/proj/toolbox1
```

Related Topics

[\\$select_reference\(\)](#)

[\\$unselect_tool\(\)](#)

[\\$unselect_config_entry\(\)](#)

[\\$unselect_version\(\)](#)

[Toolboxes](#)

\$unselect_trash_object()

Scope: dmgr_trash_area

Prerequisite: To use this function, you must be in an active Trash window and must have at least one trash object selected.

Unselects the selected or specified trash objects.

Usage

`$unselect_trash_object("name", "type")`

UNSElect TRash Object name *type*

Arguments

- **name**
A string that specifies the name of the selected trash object. You can specify the name by using UNIX System V wildcards.
- **type**
A string that specifies the type of the selected trash object. You can specify the type by using UNIX System V wildcards. The default is "", which is an empty string.
If you specify an empty string, this function unselects the first trash object whose name matches the name you specified.

Return Values

VOID.

Examples

This example unselects the "new_version" trash object.

```
$unselect_trash_object("new_version", "dme_config_do");
```

Related Topics

[\\$select_trash_object\(\)](#)

\$unselect_version()

Scope: dmgr_version_list_area

Prerequisite: To use this function, you must be in an active Versions window with at least one previously selected version.

Unselects the selected or specified version.

Usage

`$unselect_version("name", "type", version)`

UNSElect VErSION name type *version*

Arguments

- **name**
A string that specifies the name of the design object.
- **type**
A string that specifies the type of the design object.
- **version**
An integer that specifies the version number of the design object.

Return Values

VOID.

Description

By default, this function unselects the current version of the design object. To unselect a specific version, you must specify the version argument.

Examples

1. This example unselects version 3 of the "new_hampshire" design object.

```
$unselect_version("new_hampshire", "Document_file", 3);
```

2. This example unselects the same version by using command syntax from the popup command line.

```
unsel ve new_hampshire Document_file 3
```

Related Topics

[\\$select_version\(\)](#)

[\\$unselect_all\(\)](#)

[\\$unselect_by_name\(\)](#)

[\\$unselect_config_entry\(\)](#)

[\\$unselect_reference\(\)](#)

[\\$unselect_tool\(\)](#)

[\\$unselect_by_type\(\)](#)

[Versions](#)

`$unset_next_tool_env()`

Scope: `dm_tool_tk`

Removes an environment variable from the next tool environment.

Usage

```
$unset_next_tool_env("name")
```

Arguments

- **name**

A string representing the name of the environment variable to remove.

Return Values

VOID.

Description

If the environment variable already exists in the current environment, it is removed for the next invocation. If the named variable has been queued by a previous call to `$set_next_tool_env()`, a call to `$unset_next_tool_env()` removes the variable from the queue and the next tool environment.

The last call to [\\$set_next_tool_env\(\)](#) or `$unset_next_tool_env()` takes precedence.

Examples

```
$unset_next_tool_env("A")
```

Related Topics

[\\$get_next_tool_env\(\)](#)

[\\$set_next_tool_env\(\)](#)

\$untrash_object()

Scope: dmgr_trash_area

Prerequisite: To use this function, you must be in an active Trash window and must select at least one design object.

Removes the selected design object from the Trash window.

Usage

`$untrash_object("pathname")`

UNTRAsH OBject pathname

Edit > Untrash Object

Trash window popup menu > **Untrash Object**

Arguments

- **pathname**

A string that specifies the absolute pathname of the directory where this function places the trash.

Return Values

VOID.

Description

The interactive function `$untrash_object()` removes the selected design object from the Trash window and places it in the directory specified by the `pathname` argument. This function enables you to select and remove several design objects from the Trash window, simultaneously. You can also remove the selected design object from the Trash window by dragging its icon to another iconic window.

Examples

1. This example removes the selected design objects from the Trash window and places them in the `$PROJECT_XYZ/project/mail` directory.

```
$untrash_object("$PROJECT_XYZ/project/mail");
```

2. This example removes the same design objects from the trash by using command syntax from the popup command line.

```
untra ob $PROJECT_XYZ/project/mail
```

Related Topics

[\\$empty_trash\(\)](#)

[\\$trash_object\(\)](#)

\$update_objects_from_rc()

Scope: dmgr_project_model, lmrc_comp_model

Updates the specified objects with the latest version in the repository.

Usage

`$update_objects_from_rc([objects], recurse)`

Arguments

- **objects**

A vector containing a list of objects to execute with update. Each object is itself a vector containing exactly the following two strings:

- The first string is the object pathname.
- The second string is the object type.

- ***recurse***

A Boolean that specifies whether an update occurs hierarchically. Default: @true.

Return Values

VOID.

Description

This function requires that all objects reside in a managed hierarchy. Objects that have been deleted in the repository are deleted from the work area and objects that have been added to the repository are added to the work area by this function.

Objects that are unmanaged or up-to-date in the repository are skipped. Objects that are obstructing another object in the repository cannot be updated. They must first be renamed or moved so that they are no longer obstructing other objects in the repository.

Obstructed objects may be deleted.

Examples

```
$update_objects_from_rc(["$PROJECT/lib/cell", "mgc_component"]);
```

```
$update_objects_from_rc(["$PROJECT/lib/cell", "mgc_component"], ["$PROJECT/lib/cat",  
"mgc_category"]);
```

Related Topics

[\\$add_project_to_rc\(\)](#)

[\\$create_work_area_from_rc\(\)](#)

[\\$cancel_checkout_objects_for_rc\(\)](#)

[\\$checkin_objects_to_rc\(\)](#)

[\\$checkout_objects_from_rc\(\)](#)

[\\$revert_object_from_rc\(\)](#)

[\\$object_status_for_rc\(\)](#)

\$\$update_type()

Scope: dmgr_proj_tk

Updates an existing custom type, by changing the associated tool, icons, and file extensions.

Usage

```
$$update_type("type_name", "tool_name", "default_tool", "icon_string", "large_icon_string",  
             "key_name", "key_type")
```

Setup > Edit Type

Arguments

- **type_name**
A string specifying the name of the type. The type should already be in the custom type registry. New types can be created using `$$add_type()`.
- **tool_name**
A string specifying the name of the default tool for opening objects of this type.
- **default_tool**
A string specifying the path to the default tool for opening objects of this type.
- **icon_string**
A string specifying the path to the file to be used as a small icon for this type.
- **large_icon_string**
A string specifying the path to the file to be used as a large icon for this type.
- **key_name**
A string specifying the extension that is used to recognize objects of this type.
- **key_type**
A string specifying whether this type is a file type or a directory type. If the `key_type` string is "File", then files with the extension specified by `key_name` are recognized as objects of this type. If the `key_type` string is "Directory", then directories with the extension specified by `key_name` are recognized as objects of this type.

Return Values

VOID.

Description

This function requires that the user first logged in as an administrator using `$login_admin()`.

Examples

The following example updates the existing “custom_pdf” type. The type is recognized by the file extension *.pdf*. Objects of this type are displayed with the small icon at */tmp/pdf_icon*, and do not have a large icon. The custom_pdf objects are opened with */usr/bin/xpdf*.

```
$login_admin("admin1234");
```

```
$update_type("custom_pdf", "", "/usr/bin/xpdf", "/tmp/pdf_icon", "", "pdf", "File");
```

Related Topics

[\\$\\$add_type\(\)](#)

[\\$login_admin\(\)](#)

\$update_window()

Scope: dm_window

Prerequisite: To use this function, you must activate the window that you want to update.

Updates the currently active window with the most current information.

Usage

`$update_window()`

UPDate WIndow

View > Update Window

Navigator window popup menu > **Update Window**

References window popup menu > **Update Window**

Tools window popup menu > **Update Window**

Trash window popup menu > **Update Window**

Versions window popup menu > **Update Window**

Arguments

None.

Return Values

VOID.

Description

The interactive function `$update_window()` redraws the currently active window, by using the latest available information. Every time this function runs, the number of columns needed to display icons in the current window is recalculated.

Examples

1. This example updates the active window.

`$update_window();`

2. This example update the active window by using command syntax from the popup command line.

`upd wi`

\$validate_technology()

Scope: dmgr_proj_tk

Checks whether or not the target hierarchy and all its included external libraries have consistent technology settings.

Usage

```
$validate_technology("hierarchy_path")
```

Arguments

- **hierarchy_path**

A string value specifying the path to the project or external library to validate.

Return Values

A Boolean value indicating that the technology settings are valid.

Examples

The following example validates that the project at */tmp/proj* and all of its included external libraries have the same technology settings:

```
$validate_technology("/tmp/proj");
```

\$view_by_icon()

Scope: dmgr_navigator_window

Prerequisite: To use this function, you must be in an active Navigator window and must be viewing the design objects by name.

Places the current navigator in iconic-viewing mode.

Usage

`$view_by_icon()`

VIEW BY Icon

View > View by Icon

Navigator window popup menu > **View by Icon**

Arguments

None.

Return Values

VOID.

Description

The navigator must currently be in list viewing mode; otherwise, this function has no effect. Selected design objects remain selected after running this function.

Examples

1. This example places the navigator in iconic viewing mode.

`$view_by_icon();`

2. This example places the navigator in iconic viewing mode by using command syntax from the popup command line.

`vie by i`

Related Topics

[\\$open_navigator\(\)](#)

[\\$view_by_name\(\)](#)

\$view_by_name()

Scope: dmgr_navigator_window

Prerequisite: To use this function, you must be in an active Navigator window and must be viewing the design objects by icon.

Places the current navigator in list-viewing mode.

Usage

`$view_by_name()`

VIEW BY Name

View > View by Name

Navigator window popup menu > **View by Name**

Arguments

None.

Return Values

VOID.

Description

The navigator must currently be in iconic viewing mode; otherwise, this function has no effect. Selected design objects remain selected after running this function.

Examples

1. This example places the navigator in list viewing mode.

`$view_by_name();`

2. This example places the navigator in list viewing mode by using command syntax from the popup command line.

`vie by n`

Related Topics

[`\$open_navigator\(\)`](#)

[`\$view_by_icon\(\)`](#)

\$view_containment_hierarchy()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window that currently displays primaries or secondaries.

Displays the containment hierarchy of all entries in the active Configuration window, independent of their primary and secondary relationship.

Usage

`$view_containment_hierarchy()`

VIEW COntainment Hierarchy

View > Containment Hierarchy

Configuration window popup menu > **View Containment**

Arguments

None.

Return Values

VOID.

Examples

1. This example displays the containment hierarchy for the active Configuration window.

`$view_containment_hierarchy();`

2. This example displays the containment hierarchy for the active Configuration window by using command syntax from the popup command line.

`vie co h`

Related Topics

[\\$view_primary_hierarchy\(\)](#)

[\\$view_secondary_entries\(\)](#)

\$view_primary_hierarchy()

Scope: dm_config_window

Prerequisite: To use this function, you must be in an active Configuration window that currently displays the containment hierarchy.

Displays the primary entries in the active Configuration window.

Usage

`$view_primary_hierarchy()`

VIEW PRimary Hierarchy

View > Primary Hierarchy

Configuration window popup menu > **View Primaries**

Arguments

None.

Return Values

VOID.

Description

The function may show the secondary entries of each primary entry, depending on whether or not the `$view_secondary_entries()` function or the `$hid_secondary_entries()` function was run most recently. The default is to view secondary entries.

Examples

1. This example displays the primary hierarchy in the active Configuration window.

`$view_primary_hierarchy();`

2. This example display the primary hierarchy in the active Configuration window by using command syntax from the popup command line.

`vie pr h`

Related Topics

[\\$view_containment_hierarchy\(\)](#)

[\\$view_secondary_entries\(\)](#)

[\\$hide_secondary_entries\(\)](#)

\$view_secondary_entries()

Scope: dm_config_window

Prerequisite: To use this function, you must be in a Configuration window that currently hides the secondary entries.

Displays the secondary entries of a configuration in the active Configuration window.

Usage

`$view_secondary_entries()`

VIEW SEcondary Entries

View > Secondary Entries

Configuration window popup menu > **View Secondaries**

Arguments

None.

Return Values

VOID.

Examples

1. This example displays the secondary entries in the active Configuration window.

```
$view_secondary_entries();
```

2. This example displays the secondary entries in the active Configuration window by using command syntax from the popup command line.

```
vie se e
```

Related Topics

[\\$view_containment_hierarchy\(\)](#)

[\\$view_primary_hierarchy\(\)](#)

\$view_toolboxes()

Scope: dmgr_tool_window

Prerequisite: To use this function, you must be in an active Tools window.

Changes the viewing mode of the Tools window to display the current order of toolboxes in the toolbox search path.

Usage

`$view_toolboxes()`

VIEW TOOLBoxes

View > Toolboxes

Tools window popup menu > **View Toolboxes**

Arguments

None.

Return Values

VOID.

Description

The Pyxis Project Manager application searches the toolboxes found in the upper right corner of the toolbox window and then proceeds down to the next toolbox, until it reaches the lower left corner of the window.

To add a toolbox, you use the `$add_toolbox()` function. To remove a toolbox, you use the `$remove_toolbox()` function.

Examples

1. This example displays toolboxes in the toolbox search path.

```
$view_toolboxes();
```

2. This example displays toolboxes by using command syntax from the popup command line.

```
vie toolb
```

Related Topics

[\\$add_toolbox\(\)](#)

[\\$report_tool_info\(\)](#)

[\\$get_toolbox_search_path\(\)](#)

[\\$view_tools\(\)](#)

[\\$remove_toolbox\(\)](#)

\$view_tools()

Scope: dmgr_tool_window

Prerequisite: To use this function, you must be in an active Tools window that is in toolbox mode.

Changes the viewing mode of the Tools window from toolbox mode to tools mode in order to display the current set of tool icons.

Usage

`$view_tools()`

VIEW TOOLS

View > Tools

Toolboxes window popup menu > **View Tools**

Arguments

None.

Return Values

VOID.

Description

You can invoke a tool from this window by double-clicking on its icon.

Examples

1. This example displays the tool icons.

`$view_tools();`

2. This example display the tool icons by using command syntax from the popup command line.

via to

Related Topics

[\\$add_toolbox\(\)](#)

[\\$report_tool_info\(\)](#)

[\\$get_toolbox_search_path\(\)](#)

[\\$view_toolboxes\(\)](#)

[\\$remove_toolbox\(\)](#)

\$write_default_startup_file()

Scope: dmgr_session_window

Saves the values of the current session settings to the default startup file.

Usage

\$write_default_startup_file()

WRItE DEfault Startup File

Component Hierarchy window popup menu > **Write Default Startup File**

Arguments

None.

Return Values

VOID.

Description

The interactive function \$write_default_startup_file() saves the values of the current session settings to your default startup file *dmgr_default.startup*, which is located under your home directory at the path */mgc/startup*. If your default startup file already exists, the Pyxis Project Manager application asks for permission to overwrite it. If your default file does not exist, this function attempts to create it; if the Pyxis Project Manager application is unable to create the default file in the correct location, an error message is displayed in the message area stating that the Pyxis Project Manager application could not create the correct directory or file.

The \$write_default_startup_file() function saves the value of your current default editor setting, your iconic window layout, your startup window settings, your session default settings, your navigator filters, and your monitor settings.

Examples

1. This example saves your current session settings to the *dmgr_default.startup* file using function syntax.

```
$write_default_startup_file();
```

2. This example saves your current session settings to the *dmgr_default.startup* using command syntax from the popup command line.

```
wri de s f
```

Related Topics

[\\$setup_default_editor\(\)](#)

[\\$setup_session_defaults\(\)](#)

[\\$setup_iconic_window_layout\(\)](#)

[\\$setup_startup_windows\(\)](#)

\$setup_monitor()

\$\$writeln_monitor()

Scope: dm_config_tk

Writes the specified text string to the configuration's Monitor window.

Usage

`$writeln_monitor("message", type, ranking)`

Arguments

- **message**

A string containing the text that is written to the configuration monitor window.

- **type**

A name that specifies the type of the message that is contained in the message argument. The default is @plain.

- **@plain** — Prints a string to the monitor window. Whether the string is written to the monitor window is based on the verbosity setting specified by the ranking argument. Default.
- **@error** — Indicates that the string contained in the message argument is an error message. Whether the string is written to the monitor window is determined by the monitor setting currently specified for error messages.
- **@fail** — Indicates that the string contained in the message argument is a failure message. Whether the string is written to the monitor window is determined by the monitor setting currently specified for failure messages.
- **@warning** — Indicates that the string contained in the message argument is a warning message. Whether the string is written to the monitor window is determined by the monitor setting currently specified for warning messages.
- **@note** — Indicates that the string contained in the message argument is an information message, and not an error, failure, or warning message. Whether the string is written to the monitor window is determined by the monitor setting currently specified for information messages.

- **ranking**

An integer with a value in the range of 0 to 5. This argument is only valid when the type argument is specified as @plain; when the type argument is specified as something other than @plain, this argument is ignored. The integer corresponds to the verbosity setting of the configuration monitor, where a value of 1 is terse, a value of 5 is verbose, and all values in between are gradations of verbosity between the two extremes. A value of 0 indicates no ranking is applied; messages with a ranking of 0 are always printed regardless of the monitor verbosity setting. The default for this argument is 1.

Return Values

VOID.

Description

Each successive execution of this function appends the new text string to the existing text. The configuration's monitor window is automatically cleared upon each new configuration operation.

Specifying the type argument enables the message string to be filtered by the configuration monitoring settings. For example, if you specify a message string as the type warning, and your monitoring setup is set to exclude warning messages, the message is not written to the monitor area even though the `$$writeln_monitor()` function executes.

Specifying the type argument as `@plain`, which is the default value, enables you to filter messages based on a verbosity setting rather than a type setting. If you specify the type argument as `@plain` and the ranking argument as 1, you can produce a very terse monitor report without filtering out a particular type of message. If you specify the type argument as `@plain` and the ranking argument as 0, the message is always written to the monitor regardless of the configuration monitor verbosity setting.

You can customize your session setup values to specify that the configuration monitor window is either visible or hidden during configuration operations. For information about specifying your session setup values, refer to “[Changing Your Session Setup](#)” in the *Pyxis Project Manager User's Manual*.

Examples

This example demonstrates how to specify the type and ranking arguments for individual messages. If the type argument is specified as `@plain`, the message is printed if the ranking argument is less than or equal to the current monitor verbosity setting.

\$\$writeln_monitor()

```

// -----
// Defaults to @plain with ranking of 1
// -----
$$writeln_monitor("Processing Object...");
$$writeln_monitor($strcat(" Checking for: ", obj_name, ":", obj_type),
@plain, 2);
local found_it = $$object_exists(vec[i][0], vec[i][1]);
if ($$get_status_code() != 0) {
    $$monitor_global_status();
    $$writeln_monitor("Unable to continue.", @note);
    return;
}
if (found_it) {
    $$writeln_monitor($strcat(" Deleting object", @plain, 2);
    $$delete_object(obj_name, obj_type);
    if ($$get_status_code() != 0) {
        $$monitor_global_status();
        $$writeln_monitor("Unable to continue.", @note);
        return;
    }
}
// -----
// Defaults to @plain with ranking of 1
// -----
$$writeln_monitor("Process Completed");

```

Related Topics[\\$\\$clear_monitor\(\)](#)[\\$setup_monitor\(\)](#)[\\$hide_monitor\(\)](#)[\\$show_monitor\(\)](#)[\\$\\$setup_monitor\(\)](#)

Chapter 6

Integrated Pyxis Project Manager Function Dictionary

The following topics describe the Integrated Functions in the Hierarchy and Component windows.

Hierarchy Window.....	649
Component Window	672

Hierarchy Window

The table in this section lists the available functions in the context of the Hierarchy window.

Table 6-1. Summary of Hierarchy Window Functions

Function	Description
\$get_current_obj_hier_path()	Returns the instance pathname of the current object.
\$get_current_obj_inst_list()	Returns a vector of strings of the instance pathname for an instance of the current object.
\$idw_dh_setup_display()	Sets various display characteristics of the Design Hierarchy window.
\$idw_open_hierarchy_window()	Opens a new Hierarchy window on the current object.
\$idw_report_hier()	Reports the current information in the hierarchy listing.
\$inst_area_extend_selection()	Extends the selection of the instance list.
\$inst_area_select_all_items()	Selects all instances in the instance list.
\$inst_area_select_item()	Selects a single instance by name.
\$inst_area_show_instances()	Displays the next leaf of the hierarchy.
\$inst_area_unselect_all_items()	Unselects all instance in the Instance List.
\$make_obj_current()	Designates the current object.
\$open_new_comp_hierarchy()	Opens a Component Hierarchy window on a specified component.
\$open_new_hierarchy()	Opens a Design Hierarchy window on the specified component.

Table 6-1. Summary of Hierarchy Window Functions

Function	Description
\$select_obj()	Selects an object from the Hierarchy List Area.
\$set_font()	Sets the font for the Hierarchy window.
\$setup_comp_hierarchy_display()	Sets various display characteristics of the Component Hierarchy window.
\$setup_hierarchy_selection()	Sets the selection mode for the Design Hierarchy window.
\$show_instance()	Adds a leaf to the hierarchy list for the specified instance.
\$show_n_levels()	Displays the designated number of levels under the current object.

\$get_current_obj_hier_path()

Prerequisite: To use this function, you must be in an active Hierarchy window.

Returns a string that is the instance pathname of the current object.

Usage

```
$get_current_obj_hier_path (“”)
```

Arguments

None.

Return Values

A string which is the instance pathname of the current object. The current object is indicated by an outline box surrounding its name.

Examples

If you have a current object set that consists of a cardreader with the *\$PAD/newcard_reader2/card_reader/freq_det/compare/xxpls155* hierarchy, and *xxpls155* is your current object, then running this function returns the instance pathnames associated with this. For example:

```
“/I$282/I$89/I$1085”
```

Related Topics

[\\$get_current_obj_inst_list\(\)](#)

[\\$make_obj_current\(\)](#)

\$get_current_obj_inst_list()

Prerequisite: To use this function, you must be in an active Hierarchy window.

Returns a vector of strings, each of which is the instance pathname for an instance of the current object.

Usage

```
$get_current_obj_inst_list()
```

Arguments

None.

Return Values

A vector of strings, each string of which is an instance pathname for one of the instances of the current object. The vector is of the following format:

```
["instance_pathname_1", "instance_pathname_2", ..., "instance_pathname_n"]
```

Description

When multiple instance of a single object occur in the hierarchy, they are shown as a single leaf with a notation under the name indicating the number of instance the leaf represents. This builtin returns a list containing the name of each instance represented by the leaf. If the leaf represents a single instance, the vector contains only one string.

Examples

```
$get_current_obj_inst_list("$PAD/newcard_reader2/card_reader/add_det/xxo4")
```

The Return Value for the preceding example is:

```
["/$283/$1645", "/$283/$1644", "/$283/$1643", "/$283/$1641"]
```

Related Topics

[\\$get_current_obj_hier_path\(\)](#)

[\\$make_obj_current\(\)](#)

\$idw_dh_setup_display()

Prerequisite: To use this function, you must be in an active Hierarchy window.

Sets various display characteristics in the Design Hierarchy window.

Usage

```
$idw_dh_setup_display(tree_display_type, path_type, instance_list_visibility,  
    instance_display_type, "property_name", instance_name_visibility, model_visibility)
```

Component Hierarchy window popup menu > **Setup** > **Display**

Arguments

- **tree_display_type**

A name that indicates the type of tree structure used to display the hierarchy. Options include:

- **@listing** — Produces an indented list type.
- **@right** — Produces a graphical tree shown left to right. You can override this default at startup using a startup file. Default.
- **@down** — Produces a graphical tree shown top to bottom.

- **path_type**

A name that indicates whether full pathnames or leaf name are used to display components in the hierarchy.

- **@full_path** — Displays the full pathname for the instance.
- **@leaf** — Displays only the leaf name of the instance. You can override this default at startup using a startup file. Default.

- **instance_list_visibility**

A name that indicates whether to hide or display the instance list.

- **@hide_instance_list** — Hides the instance list from view.
- **@show_instance_list** — Shows the instance list. You can override this default at startup using a startup file. Default.

- **instance_display_type**

A name that specifies whether instance names or a property value should be shown next to component names. This argument specifies only the kind of information shown in parenthesis next to components (instance name or property value). Another argument, `$instance_name_visibility()`, tells whether to show the information or not (an on/off toggle).

- **@instance_name** — The instance names appear in the hierarchy list in parentheses next to component names when the argument `instance_name_visibility` is appropriately set.

- **@property_name** — A specified properties' value is shown rather than an instance name.
- **property_name**
A string that is a property name. The property name is only used if \$instance_display_type() is set to @property_value, and if \$instance_name_visibility() is set to @show_instance_names. The default is "", but is overridden by whatever is stored in the startup files.
- **instance_name_visibility**
A name used to specify whether additional information should be displayed in parenthesis next to component names in the hierarchy listing. This acts as an on/off switch for the information. The type of information shown in parenthesis is specified by the \$instance_display_type() argument, and may be either an instance name or a property value.
 - **@show_instance_names** — Shows the instance names in parenthesis next to the component. You can override this default at startup using a startup file. Default.
 - **@hide_instance_names** — Hides any instance names next to the component.
- **model_visibility**
A name used to specify whether model information is displayed next to component names in the hierarchy listing. This acts as an on/off switch for the information. The model information consists of a small type icon, and model name.
 - **@hide_models** — Hides the model information. You can override this default at startup using a startup file. Default.
 - **@show_models** — Displays the model information.

Return Values

VOID.

Related Topics

[\\$idw_open_hierarchy_window\(\)](#)

[\\$setup_comp_hierarchy_display\(\)](#)

[\\$idw_report_hier\(\)](#)

[\\$setup_hierarchy_selection\(\)](#)

\$idw_report_hier()

Prerequisite: To use this function, you must be in an active Hierarchy window.

Creates a report window containing the current (non-graphical) information in the hierarchy listing.

Usage

```
$idw_report_hier(“”)
```

Component Hierarchy window popup menu > **Report**

Arguments

None.

Return Values

VOID.

Description

The hierarchy is presented as an indented list format regardless of the actual format in the Hierarchy window. The report window has its own popup menu that contains items used to export the information to an ASCII file or printer.

Examples

```
$idw_report_hier(“”)
```

Related Topics

[\\$idw_dh_setup_display\(\)](#)

[\\$setup_hierarchy_selection\(\)](#)

[\\$idw_open_hierarchy_window\(\)](#)

\$idw_open_hierarchy_window()

Prerequisite: To use this function, you must be in an active Hierarchy window.

Opens either a Design Hierarchy window or a Component Hierarchy window depending on the application from where it is used.

Usage

```
$idw_open_hierarchy_window("instance_path", type)
```

Arguments

- **instance_path**

A string that represents the top of the hierarchy. If set to "/" the design shown is the design specified by the currently opened viewpoint in the application. This is the normal case when a Hierarchy window is first opened in an application.

If set to an instance path, such as that returned by \$get_current_obj_hier_path(), then the Hierarchy window shows a design whose top level is the specified instance.

- **type**

A name value which is either @current_viewpoint or @path.

- **@current_viewpoint** — The Hierarchy window is opened using the currently opened viewpoint, and the instance_path argument is ignored. This results in an evaluated design hierarchy. Default.
- **@path** — The hierarchy window opens on the component specified in the instance_path argument, and is a Component Hierarchy window rather than a Design Hierarchy window.

Return Values

VOID.

Description

The hierarchy that is displayed starts from the root of the design (instance path =/) or from the point specified by the instance_path argument.

This wrapper function determines the data type when a path is given. EDDM, IC and SDS application types are supported and when displayed are noted in the banner of the Hierarchy window.

Examples

```
$idw_open_hierarchy_window("", @current_viewpoint)
```

```
$idw_open_hierarchy_window("<pathname>", @path)
```


Related Topics

[`\$idw_dh_setup_display\(\)`](#)

[`\$setup_hierarchy_selection\(\)`](#)

[`\$idw_report_hier\(\)`](#)

\$inst_area_extend_selection()

Prerequisite: To use this function, you must be in an active Hierarchy window.

Operates in the instance list to extend the selection to include all instance between the last selected instance up to and including the specified instance.

Usage

```
$inst_area_extend_selection(“”)
```

Arguments

- **instance_name**
A string that specifies an instance name.

Return Values

VOID.

Examples

The following command extends the selection:

```
$inst_area_extend_selection("I$2891")
```

Related Topics

[\\$inst_area_select_all_items\(\)](#)

[\\$inst_area_unselect_all_items\(\)](#)

[\\$inst_area_select_item\(\)](#)

[\\$inst_area_show_instances\(\)](#)

\$inst_area_select_all_items()

Prerequisite: To use this function, you must be in an active Hierarchy window.

Selects all instances in the instance list area.

Usage

`$inst_area_select_all_items()`

Arguments

None.

Return Values

VOID.

Description

Instances shown in the Instance list are for the current object in the Hierarchy List. The current object is indicated by an outline box. The current object is set to the last item selected, and it can be moved using the keyboard arrow keys or the left mouse button.

The Graphical User Interface (GUI) provides access to this builtin through the Instance List popup menu item **Select > List**.

Examples

This item is defined as follows:

```
$menu_text_item("_Select List","$inst_area_select_all_items()")
```

Related Topics

[\\$inst_area_extend_selection\(\)](#)

[\\$inst_area_unselect_all_items\(\)](#)

[\\$inst_area_select_item\(\)](#)

[\\$inst_area_show_instances\(\)](#)

\$inst_area_select_item()

Prerequisite: To use this function, you must be in an active Hierarchy window.

Selects a single instance by name from the Instance List and adds the instance to the selected set.

Usage

```
$inst_area_select_item("instance_name")
```

Arguments

None.

Return Values

VOID.

Examples

```
$inst_area_select_item("$2891")
```

Related Topics

[\\$inst_area_extend_selection\(\)](#)

[\\$inst_area_unselect_all_items\(\)](#)

[\\$inst_area_select_all_items\(\)](#)

[\\$inst_area_show_instances\(\)](#)

\$inst_area_show_instances()

Prerequisite: To use this function, you must be in an active Hierarchy window.

Adds a leaf to the hierarchy listing for each instance selected in the Instance List.

Usage

\$inst_area_show_instances()

Arguments

None; operates on what is selected in the Instance List.

Return Values

VOID.

Description

When more than one instance exists for a component, a single leaf is used to represent the multiple instance. The multiple instances are represented by a single leaf that notes, in parenthesis beneath the component name, how many additional instances are represented by the leaf. The individual instance names appear in the Instance List when the leaf is made current.

Leaves may be added to the hierarchy for one or more of the additional instances by first selecting them in the Instance List, and then issuing \$inst_area_show_instances(). The GUI provides access to this command from the Instance List popup menu item **Show Instances**.

Related Topics

[\\$inst_area_extend_selection\(\)](#)

[\\$inst_area_unselect_all_items\(\)](#)

[\\$inst_area_select_all_items\(\)](#)

[\\$show_instance\(\)](#)

[\\$inst_area_select_item\(\)](#)

\$inst_area_unselect_all_items()

Prerequisite: To use this function, you must be in an active Hierarchy window.

Unselects all instances in the Instance List.

Usage

`$inst_area_unselect_all_items()`

Arguments

None.

Return Values

VOID.

Description

Instances that are shown in the Instance List represent only those associated with the current object. The current object is designated by the outline box in the Hierarchy List Area and is set to the last object selected.

The GUI provides access to this builtin through the Instance List popup menu item **Unselect List**. T

Examples

his item is defined as follows:

```
$menu_text_item ("_Unselect List", "$inst_area_unselect_all_items()")
```

Related Topics

[\\$inst_area_extend_selection\(\)](#)

[\\$inst_area_select_item\(\)](#)

[\\$inst_area_select_all_items\(\)](#)

[\\$inst_area_show_instances\(\)](#)

\$make_obj_current()

Prerequisite: To use this function, you must be in an active Hierarchy window.

Designates the specified object as the current one.

Usage

`$make_obj_current("object_instance_pathname", keep_visible)`

Arguments

- **object_instance_pathname**
A string that is an object's instance pathname.
- ***keep_visible***
A Boolean specifying whether to move the view of the Hierarchy List so that the current object is visible in the window.
 - **@true** — Moves the view to show the current object.
 - **@false** — Does not move the view to show the current object. Default.

Return Values

VOID.

Description

The current object is indicated by an outline box around its name. The current object is one whose instances are shown in the Instance List. There is only one current object at any one time. This is in contrast to selected objects, of which there can be more than one. When there is more than one object selected the current object is by default the last object selected.

Related Topics

[\\$get_current_obj_hier_path\(\)](#)

[\\$select_obj\(\)](#)

[\\$get_current_obj_inst_list\(\)](#)

\$open_new_comp_hierarchy()

Prerequisite: To use this function, you must be in an active Component Hierarchy window.

Opens a Component Hierarchy window on the component specified by the input object information.

Usage

`$open_new_comp_hierarchy([obj])`

Component Hierarchy window popup menu > **Open New Hierarchy**

Arguments

- **obj**

A vector containing information about the current object. The vector is of the format: ["object_pathname", "object_type", "model_pathname", "model_type"], which is the same as that returned by `$get_current_obj_info()`.

Return Values

VOID.

Description

This is a the interactive function wrapper used to open another Hierarchy window from within an existing component hierarchy window.

The GUI seeds this wrapper function with the current component's information, so that a new Hierarchy window is automatically opened on the current component. The current component is designated by an outline box and is usually the last object selected.

A similar command, `$open_new_hierarchy()`, is used to open a design hierarchy window from an existing Design Hierarchy window.

Examples

`$open_new_comp_hierarchy($get_current_obj_info())`

Related Topics

[\\$open_new_hierarchy\(\)](#)

\$open_new_hierarchy()

Prerequisite: To use this function, you must be in an active Design Hierarchy window.

Opens a Design Hierarchy window on the component specified by the input object information.

Usage

```
$open_new_hierarchy("instance_pathname")
```

Arguments

- **instance_pathname**

A string that is the instance pathname to the component from which a new Hierarchy window is opened.

Return Values

VOID.

Description

This is the interactive function wrapper used to open another Hierarchy window from within an existing Design Hierarchy window. The GUI seeds this wrapper function with the current component's instance pathname, so that a new Hierarchy window is automatically opened on the current component. The current component is designated by an outline box and is usually the last object selected. You can designate a different instance pathname then that which is seeded into form if required.

A similar command, `$open_new_comp_hierarchy()`, is used to open a Component Hierarchy window from an existing Component Hierarchy window.

Examples

An example instance path string:

```
$open_new_hierarchy("/I$283/I$686")
```

Related Topics

[\\$open_new_comp_hierarchy\(\)](#)

\$select_obj()

Prerequisite: To use this function, you must be in an active Hierarchy window.

Selects an object from the Hierarchy list area.

Usage

`$select_obj("object_instance_pathname")`

Arguments

- **object_instance_pathname**
A string that is an object's instance pathname.

Return Values

VOID.

Description

The object selected becomes the current object.

The `$select_obj()` command does not apply to windows that do not support selected objects, such as the Component Hierarchy window.

Examples

`$select_object("/I$282/I$88/I$543")`

Related Topics

[\\$make_obj_current\(\)](#)

\$show_instance()

Prerequisite: To use this function, you must be in an active Hierarchy window.

Adds a leaf to the hierarchy list for the specified instance.

Usage

```
$show_instance("instance_path")
```

Arguments

- **instance_path**
A string that is an instance pathname.

Return Values

VOID.

Description

When multiple instance of a single object occur in the hierarchy, they are shown as a single leaf with a notation under the component name indicating the number of instances the leaf represents. This builtin can be used to add a leaf for specified instance.

Examples

```
$show_instance("/I$282/I$89/I$1085/I$573")
```

Related Topics

[\\$inst_area_show_instances\(\)](#)

\$show_n_levels()

Prerequisite: To use this function, you must be in an active Hierarchy window.

Updates the hierarchy list to show components underneath the current object to the specified hierarchical depth.

Usage

`$show_n_levels("object_hier_path", number_of_levels)`

Component Hierarchy window popup menu > **Show Levels** > **Show n Levels**

Arguments

- **object_hier_path**
A string that is an object's instance pathname
- **number_of_levels**
An integer that is the number of additional hierarchical levels to show below the specified object. If set to -1, then all levels below the specified object are shown.

Return Values

VOID.

Examples

`$show_n_levels("/I$282", 2)`

\$set_font()

Prerequisite: To use this function, you must be in an active Hierarchy window.

Sets the font used to display text in the Hierarchy window pane.

Usage

```
$set_font("font_name")
```

Component Hierarchy window popup menu > **Setup** > **Set Font**

Arguments

- **font_name**
A string that is a font name.

Return Values

VOID.

Description

Selecting the font name changes the font of text in the Hierarchy window to selected font style.

Examples

```
$set_font("times-bold:18")
```

\$setup_comp_hierarchy_display()

Prerequisite: To use this function, you must be in an active Hierarchy window.

Sets various display characteristics in the Component Hierarchy window .

Usage

`$setup_comp_hierarchy_display(tree_display_type, path_type)`

Component Hierarchy window popup menu > **Setup** > **Display**

Arguments

- **tree_display_type**

A name that indicates the type of tree structure used to display the hierarchy.

- **@listing** — Produces an indented list type.
- **@right** — Produces a graphical tree shown left to right. You can override this default at startup using a startup file. Default.
- **@down** — Produces a graphical tree shown top to bottom.

- **path_type**

A name that indicates whether full pathnames or leaf name are used to display components in the hierarchy.

- **@full_path** — Displays the full pathname for the instance.
- **@leaf** — Displays only the leaf name of the instance. You can override this default at startup using a startup file. Default.

Return Values

VOID.

Related Topics

[\\$idw_dh_setup_display\(\)](#)

\$setup_hierarchy_selection()

Prerequisite: To use this function, you must be in an active Hierarchy window.

Sets the selection mode for the Design Hierarchy window.

Usage

```
$setup_hierarchy_selection(mode)
```

Arguments

- **mode**

Options include:

- **@all** — Selects all instance when selecting a component leaf that represents multiple instances.
- **@one** — Selects only the first instance when selecting a component leaf which represents multiple instances.
- **@none** — Disable selection. Do not select instance when clicking on a component leaf.

Return Values

VOID.

Examples

```
$setup_hierarchy_selection(@all)
```

Related Topics

[\\$idw_dh_setup_display\(\)](#)

[\\$setup_comp_hierarchy_display\(\)](#)

[\\$idw_report_hier\(\)](#)

Component Window

The table in this section lists the available functions in the context of the Component window.

Table 6-2. iDM Component Window Functions

Function	Description
\$add_components()	Adds one or more components to the Component Information List.
\$add_labels_to_models()	Adds the specified labels to all specified models.
\$collapse_object()	Collapses objects shown as being contained by the specified object of the specified type.
\$delete_labels_from_models()	Deletes one or more model labels.
\$delete_part_interfaces()	Deletes one or more part interfaces.
\$expand_object()	Expands the Component Information List display to include objects underneath the specified object type.
\$forget_components_edits()	Changes the status of one or more specified components.
\$hide_body_props()	Hides the Body Properties List.
\$hide_labels()	Hides labels in the Registered Models List.
\$hide_model()	Hides the Model List.
\$hide_pin_properties()	Hides pin properties in the Pins List.
\$hide_pins()	Hides the Pins List.
\$register_models()	Registers one or more models to the part interface.
\$remove_components()	Removes one or more components from the Component Information List.
\$rename_part_interface()	Renames the part interface.
\$report_body_prop_info()	Creates a report window containing the Body Properties List.
\$report_component_info()	Creates a report window containing the Component Information List.
\$report_model_entry_info()	Creates a report window containing model validity information.
\$report_models_for_each_label()	Creates a report window containing model information for specified label.
\$report_model_info()	Creates a report window containing the Registered Model Information List.

Table 6-2. iDM Component Window Functions (cont.)

Function	Description
\$report_models_with_all_labels()	Creates a report window containing the information on the models that all have a specified label.
\$report_pin_info()	Creates a report window containing the Pins List.
\$save_components_edits()	Saves edits made to components.
\$select_model_object()	Selects a model that is currently shown in the Registered Model Information List.
\$select_object()	Selects an object or objects in the Component Information List.
\$set_bgd_color()	Sets the color for the list backgrounds.
\$set_bgd_color_title_items()	Sets the color for the background labels bars.
\$set_bgd_color_titles()	Sets the color for the list title bars.
\$set_constraints()	Sets the list area width and orientation.
\$set_default_part_interface()	Sets the default part interface.
\$set_fgd_color()	Sets the color for the foreground (list text).
\$set_fgd_color_title_items()	Sets the color for the foreground title bars.
\$set_fgd_color_titles()	Sets the color for the foreground list title bar.
\$set_font()	Sets the font to use to display text.
\$set_part_interface_font()	Sets the font to use to display part interface names.
\$show_body_props()	Displays the Body Properties List.
\$show_labels()	Displays the labels shown underneath the models.
\$show_model()	Displays the Registered Model Information List.
\$show_pins()	Displays the Pins List.
\$show_pin_properties()	Displays the pin properties underneath the pin in the Pin List.
\$unselect_model_object()	Unselects an object in the Registered Model Information List.
\$unselect_object()	Unselects an object in the Component Information List.
\$validate_models()	Performs a validity check on specified models.

\$add_components()

Prerequisite: To use this function, you must be in an active Component window.

Adds the specified components to the Component Information pane in the Component window.

Usage

`$add_components([components])`

Component window > **Component Information** pane popup menu > **Add Components**

Arguments

- **components**

A vector of the strings which are the individual component pathnames. The vector format is:

[“component_1”, “component_2”, “component_3”]

Return Values

VOID.

Description

Only objects that are of the “mgc_component” type are added to the Component Information pane.

Examples

This example adds the */user/in_process/perf_ckt* component to the Component Information pane:

```
$add_components("/user/in_process/perf_ckt)
```

Related Topics

[\\$forget_components_edits\(\)](#)

[\\$report_component_info\(\)](#)

[\\$remove_components\(\)](#)

[\\$save_components_edits\(\)](#)

\$add_labels_to_models()

Prerequisite: To use this function, you must be in an active Component window.

Adds the specified labels to the specified models in the Registered Model Info pane of the Component window.

Usage

`$add_labels_to_models([label_names], [models])`

Component window > **Registered Model Info** pane popup menu > **Edit** > **Add Labels**

Arguments

- **label_names**

A vector of strings where each string is a label name.

- **models**

A vector of vectors. Each sub vector describes a model. The models argument can contain any number of sub-vector model descriptions. Each sub-vector contains four string elements in the following format:

`["component_pathname", "part_interface_name", "model_name", "model_type"]`

So that the models argument is in general of the format:

`[["comp_1", "pi_1", "model_1", "type_1"], ["comp_2", "pi_2", "model_2", "type_2"], ...]`

Return Values

VOID.

Description

This function can be used to add one or more labels to one or more models. Each label is added to each model.

The Models list popup menu item **Add Labels** provides an interface to this builtin which prompts for label names, and then adds labels to all models currently selected in the Model list.

Examples

The following adds two labels to two different models. The models are in two different components:

```
$add_labels_to_models(  
  ["TYPE", "default_sym"],  
  ["/user/in_process/741s74a", "MGC_STD", "/users/in_process/741s74a/MG_STD",  
    "mgc_symbol"],  
  
  ["/user/in_process/card_reader", "card_reader", "/users/in_process/card_reader/card_re  
ader", "mgc_symbol"]])
```

Related Topics

[\\$delete_labels_from_models\(\)](#)

[\\$hide_labels\(\)](#)

[\\$report_models_for_each_label\(\)](#)

[\\$report_models_with_all_labels\(\)](#)

[\\$show_labels\(\)](#)

\$collapse_object()

Prerequisite: To use this function, you must be in an active Component window.

Collapses objects shown as being contained by the specified object of the specified type.

Usage

`$collapse_object(type, object)`

Arguments

- **type**
A string representing an object type (e.g. “mgc_component” or “mgc_schematic”). The argument may be wildcarded. For example, possible values for the type argument are:
 - “*” — To represent models of any type.
 - “mgc*” — To represent all models whose type name begins with “mgc”.
- **object**
A string that represents the object pathname. This argument may also be wildcarded. For example, possible values of the objects argument are:
 - “*” — To represent objects with any name (all objects).
 - “*schematic*” — To represent any object whose name includes the string “schematic”.
 - “/user/in_process/741s74a” — To represent the object 741s74a only.

Return Values

VOID.

Description

Each argument may be wild-carded; for example, all objects of the “mgc_component” type can be collapsed at one time. This command affects the display only and does not change data in memory or on disk.

The Left Mouse Button 2 function is defined to toggle on/off the appearance of information underneath a component. The `$collapse_object()` and `$expand_object()` functions are transcribed depending on the action taken.

Examples

```
$collapse_object("", "*analysis")
```

Related Topics

[\\$expand_object\(\)](#)

\$delete_labels_from_models()

Prerequisite: To use this function, you must be in an active Component window.

Deletes one or more model labels selected in the list of models in the Component window.

Usage

`$delete_labels_from_models([labels])`

Component window > **Registered Model Info** pane popup menu > **Edit** > **Delete Labels**

Arguments

- **labels**

A vector of one or more sub-vectors. Each sub-vector describes a label to be deleted. The sub-vector contains five string elements in the following format:

```
[“component_pathname”, “part_interface”, “model_name”, “model_type”,  
label_name”]
```

The labels argument can contain any number of these sub-vectors, so that any number of labels can be selected at one time. Wild-carding is enabled on the sub-vectors.

Return Values

VOID.

Examples

The following command deletes the label TYPE from the model MG_STD in the component 741s74a's part interface called MGC_STD.

```
$delete_labels_form_models([["/users/in_process/741s74a",  
"MG_STD", "/users/in_process/741s74a/MGC_STD",  
"mgc_symbol", "TYPE"]])
```

Related Topics

[\\$add_labels_to_models\(\)](#)

[\\$report_models_with_all_labels\(\)](#)

[\\$hide_labels\(\)](#)

[\\$show_labels\(\)](#)

[\\$report_models_for_each_label\(\)](#)

\$delete_part_interfaces()

Prerequisite: To use this function, you must be in an active Component window.

Deletes one or more part interfaces.

Usage

`$delete_part_interfaces([part_interface])`

Component window > **Component Information** pane popup menu > **Edit** > **Delete Part Interfaces**

Arguments

- **part_interface**

A vector of one or more sub-vectors. Each sub-vector describes a part interface to be deleted. The sub-vector contains 2 string elements in the following format:

`["component_pathname", "part_interface"]`

The **component_pathname** argument can contain one or more of the sub-vectors. The **component_pathname** item can be wildcarded.

Return Values

VOID.

Description

A part interface must have an empty model table (where no models are registered to the part interface) for it to be deleted.

Examples

1. The following command deletes the part interface OLD from the component 741s74a. Note the double brackets as the argument is a vector of vectors (even though, in the example, only one part interface is specified).

```
$delete_part_interfaces([["/user/s/davidr/comp/741s74a", "OLD"]])
```

2. The following command deletes the part interface named MG_STD from any component which has a part interface of that name.

```
$delete_part_interfaces([["*", "OLD"]])
```

Related Topics

[\\$rename_part_interface\(\)](#)

[\\$set_part_interface_font\(\)](#)

[\\$set_default_part_interface\(\)](#)

\$expand_object()

Prerequisite: To use this function, you must be in an active Component window.

Expands the display of the Component Information pane in the Component window to include objects underneath (or contained) by the specified object of the specified type.

Usage

`$expand_object("type", "object")`

Arguments

- **type**
A string representing an object type (for example, "mgc_component" or "mgc_schematic"). The argument may be wildcarded. For example, possible values for the type argument are:
 - "*" — To represent models of any type.
 - "mgc*" — To represent all models whose type name begins with "mgc".
- **object**
A string that represents the object pathname. This argument may be wildcarded. For example, possible values for the objects argument are:
 - "*" — To represent objects with any name (all objects).
 - "*schematic*" — To represent any object whose name includes the string "schematic".
 - "/user/in_process/741s74a" — To represent the object 741s74a only.

Return Values

VOID.

Description

This command affects the display only, and does not change data in memory or on disk. The \$expand_object() function only operates on objects that are already displayed in the Component Information pane.

The Left Mouse Button 2 function is defined to toggle on/off the appearance of information underneath a component. The functions \$collapse_object() and \$expand_object() are transcribed depending on the action taken (whether the contained information is hidden or shown).

Examples

`$expand_object("*", "*analysis")`

Related Topics

[\\$collapse_object\(\)](#)

\$forget_components_edits()

Prerequisite: To use this function, you must be in an active Component window.

Changes the state of one or more specified components from changed to unchanged, without first saving the changes to disk.

Usage

`$forget_components_edits([components])`

Component window > **Component Information** pane popup menu > **Edit** > **Forget/Update Selected Component Edits**

Arguments

- **components**

A vector containing one or more strings, each of which is a component pathname. The format is:

["component_pathname_1", "component_pathname_2"...
"component_pathname_n"]

Return Values

VOID.

Description

Once forgotten, edits are not recoverable except by re-editing the component.

Once an edit occurs to a component, that is adding a model label, registering a model and so on, then the component is in the changed state until the changes are saved using the `$save_components_edits()` function or forgotten using the `$forget_components_edits()` function. Components in the changed state are distinguished by the background color of the icon being white.

Examples

`$forget_components_edits(["users/in_process/741s74a"])`

Related Topics

[\\$add_components\(\)](#)

[\\$report_component_info\(\)](#)

[\\$remove_components\(\)](#)

[\\$save_components_edits\(\)](#)

\$hide_body_props()

Prerequisite: To use this function, you must be in an active Component window and the Body Properties pane must be displayed.

Hides the Body Properties pane in the Component window.

Usage

\$hide_body_props()

Component window > **Body Properties** pane popup menu > **Hide Body Properties List**

Arguments

None.

Return Values

VOID.

Description

When the Body Properties pane is hidden, other lists become larger to fill the window area that is available. Issuing this builtin when the Body Properties pane is already hidden has no effect.

The Body Properties pane can be shown or hidden using menu items from the popup menu in both the Component Information pane and the Body Properties pane.

Examples

\$hide_body_props()

Related Topics

[\\$report_body_prop_info\(\)](#)

[\\$show_body_props\(\)](#)

\$hide_labels()

Prerequisite: To use this function, you must be in an active Component window and the labels must be shown in the Registered Model Info pane.

Hides the display of labels in the Registered Model Info pane of the Component window.

Usage

\$hide_labels()

Component window > **Registered Model Info** pane popup menu > **Hide Labels**

Arguments

None.

Return Values

VOID.

Description

Issuing this builtin when the labels are already hidden has no effect.

Labels in the Registered Model Info pane can be shown or hidden using the popup menu, which contains an item that toggles the visibility on/off. The item changes between Show Labels and Hide Labels, and the menu items use the builtins \$show_labels() and \$hide_labels() respectively.

Examples

\$hide_labels()

Related Topics

[\\$add_labels_to_models\(\)](#)

[\\$report_models_with_all_labels\(\)](#)

[\\$delete_labels_from_models\(\)](#)

[\\$show_labels\(\)](#)

[\\$report_models_for_each_label\(\)](#)

\$hide_model()

Prerequisite: To use this function, you must be in an active Component window and the Registered Model Info pane must be visible.

Hides the Registered Model Info pane in the Component window.

Usage

\$hide_model()

Component window > **Registered Model Info** pane popup menu > **Hide Models List**

Description

When the Registered Model Info pane is hidden, other lists become larger to fill the window area that is available. Issuing this builtin when the Registered Model Info pane is already hidden has no effect.

The Registered Model Info pane can be shown or hidden using menu items from the popup menus in the Component Information pane and the Registered Model Info pane.

Arguments

None.

Return Values

VOID.

Examples

\$hide_model()

Related Topics

[\\$show_model\(\)](#)

\$hide_pin_properties()

Prerequisite: To use this function, you must be in an active Component window and the pin properties in the Pins pane must be visible.

Hides the pin properties that may be showing in the Pins pane of the Component window.

Usage

\$hide_pin_properties()

Component window > **Pins** pane popup menu > **Hide Pin Properties**

Arguments

None.

Return Values

VOID.

Description

Issuing this builtin when the pin properties are already hidden has no effect.

Pin properties in the Pins pane can be shown or hidden using the popup menu, which contains an item that toggles the visibility on/off. The item changes between Show Pin Properties and Hide Pin Properties, and the menu items use the builtins \$show_pin_properties() and \$hide_pin_properties() respectively.

Examples

\$hide_pin_properties()

Related Topics

[\\$show_pin_properties\(\)](#)

\$hide_pins()

Prerequisite: To use this function, you must be in an active Component window and the Pins pane must be visible.

Hides the Pins pane of the Component window.

Usage

\$hide_pins()

Component window > **Pins** pane popup menu > **Hide Pins List**

Description

When the Pins pane is hidden, other lists become larger to fill the void. Issuing this builtin when the Pins pane is already hidden has no effect.

The Pins pane can be shown or hidden using menu items from the popup menus in the Component Information pane and the Pins pane.

Arguments

None.

Return Values

VOID.

Examples

\$hide_pins()

Related Topics

[\\$report_pin_info\(\)](#)

[\\$show_pins\(\)](#)

\$register_models()

Prerequisite: To use this function, you must be in an active Component window.

Registers one or more models to one or more part interfaces.

Usage

`$register_models([models])`

Component window > **Component Information** pane popup menu > **Edit** > **Register Model**

Arguments

- **models**

A vector of vectors, each sub-vector of which describes a model and part interface. The sub-vector contains four strings in the following format:

["component_pathname", "part_interface_name", "model_name", "model_type"]

The vector can contain any number of these sub-vectors separated by commas, thus enabling the registration of more than one model at a time.

Return Values

VOID.

Description

The "models" argument is a vector of vectors, each sub-vector of which describes a model to be registered, and the part interface with which it is being registered. Only models of type *mgc_schematic* or *mgc_symbol* may be registered.

The user interface provides a wrapper function to this builtin. The function operates off of the Component Information pane, and registers selected models with selected part interfaces. A form appears, which lists all the selected models and part interfaces, and asks for confirmation that all models be registered with all part interfaces.

Examples

The following example registers two models with two part interfaces from two different components. The models MG_STD and schematic, of type *mgc_symbol* and *mgc_schematic* respectively, are added to the part interfaces MG_STD from the component 7rls74a and to the part interface card_reader from the component called card_reader.

```
$register_models ([["/user/in_process/741s74a", "MG_STD",  
                  "/users/in_process/741s74a/MG_STD", "mgc_symbol"],  
                  ["/users/in_process/741s74a", "MG_STD",  
                  "/users/in_process/newcard_reader2/card_reader/schematic", "mgc_schematic"],  
                  ["/users/in_process/newcard_reader 2/card_reader", "card_reader",  
                  "/users/in_process/741s74a/MG_STD", "mgc_symbol"],  
                  ["/users/in_process/newcard_reader2/card_reader", "card_reader",  
                  "/users/in_process/newcard_reader2/card_reader/schematic", "mgc_schematic"]]);
```


Related Topics

[`\$report_model_entry_info\(\)`](#)

[`\$validate_models\(\)`](#)

\$remove_components()

Prerequisite: To use this function, you must be in an active Component window.

Removes one or more components from the Component Information pane of the Component window.

Usage

`$remove_components([components])`

Component window > **Component Information** pane popup menu >
Remove Selected Components

Arguments

- **components**

A vector of strings which are the individual component pathnames. The vector format is:

["component_pathname_1", "component_pathname_2", ...,
"component_pathname_n"]

Return Values

VOID.

Description

Only components which have been added to the Component window using the `$add_components()` function or those that were specified when the Component window was first opened can be removed. Components which are contained (or shown underneath) other components cannot be removed (except by removing their parent component).

The user interface provides a wrapper function to this builtin which removes the selected components from the Component Information pane. The function is available from the popup menu of the Component Information pane.

When a component is removed from the Component window, only the display is updated. No changes are made to the component on disk.

Examples

`$remove_components(["/user/in_process/perf_ckt"])`

Related Topics

[\\$add_components\(\)](#)

[\\$report_component_info\(\)](#)

[\\$forget_components_edits\(\)](#)

[\\$save_components_edits\(\)](#)

\$rename_part_interface()

Prerequisite: To use this function, you must be in an active Component window.

Renames the specified part interface in the specified component(s).

Usage

`$rename_part_interface("component_pathname", "old_pi_name", "new_pi_name")`

Component window > **Component Information** pane popup menu > **Edit** >
Rename Part Interface

Arguments

- **component_pathname**
A string that is a component pathname. The string may be wildcarded ("*" means all components).
- **old_pi_name**
A string that is the name of an existing part interface.
- **new_pi_name**
A string that becomes the new name of the part interface.

Return Values

VOID.

Examples

`$rename_part_interface("/users/stevenb/comp/741s74a", "OLD", "CURRENT")`

Related Topics

[\\$delete_part_interfaces\(\)](#)

[\\$set_part_interface_font\(\)](#)

[\\$set_default_part_interface\(\)](#)

\$report_body_prop_info()

Prerequisite: To use this function, you must be in an active Component window and the Body Properties pane must be displayed.

Creates a report window containing information displayed in the Body Properties pane of the Component window.

Usage

\$report_body_prop_info()

Component window > **Body Properties** pane popup menu > **Report List Area**

Arguments

None.

Return Values

VOID.

Description

The report window contains its own popup menu with items for exporting the information to a printer file.

Examples

\$report_body_prop_info()

Related Topics

[\\$hide_body_props\(\)](#)

[\\$show_body_props\(\)](#)

\$report_component_info()

Prerequisite: To use this function, you must be in an active Component window and have a component selected.

Creates a report window containing information displayed in the Component Information pane of the Component window.

Usage

\$report_component_info()

Component window > **Component Information** pane popup menu > **Report List Area**

Arguments

None.

Return Values

VOID.

Description

The report window contains its own popup menu with items for exporting the information to a printer or file.

Examples

\$report_component_info()

Related Topics

[\\$add_components\(\)](#)

[\\$remove_components\(\)](#)

[\\$forget_components_edits\(\)](#)

[\\$save_components_edits\(\)](#)

\$report_model_entry_info()

Prerequisite: To use this function, you must be in the Registered Model Info pane of the Component window and must select a model in the list.

Creates a report window containing information on the validity of specified models.

Usage

`$report_model_entry_info([models], report_window)`

Component window > **Registered Model Info** pane popup menu > **Report** > **Model Validity**

Arguments

- **models**

A vector which describes models. The vector is actually a vector of one or more sub-vectors. Each sub-vector describes one model in the format:

[component_pathname, part_interface, model_pathname, model_type]

The model vector is the same format as that returned by `$get_selected_models()`.

- **report_window**

A switch that specifies whether or not to create a report window.

- **@report** — Creates a report window. Default.
- **@noreport** — Reports validity of models as transcribed messages.

Return Values

VOID.

Examples

`$report_model_entry_info()`

Related Topics

[\\$register_models\(\)](#)

[\\$validate_models\(\)](#)

\$report_models_for_each_label()

Prerequisite: To use this function, you must be in an active Component window.

Produces a report window containing information on the models that have a specified label.

Usage

`$report_models_for_each_label([labels], [part_interfaces])`

Component window > **Registered Model Info** pane popup menu > **Report** > **Models By Each Label**

Arguments

- **labels**

A vector containing one or more strings, each of which is a label name.

- **part_interfaces**

A vector containing information about one or more part interfaces. Each subvector contains two strings describing a component and part interface. The format of the subvector is:

["component_pathname", "part_interface"]

Return Values

VOID.

Description

Only models from the specified part interface are considered.

Examples

`$report_models_for_each_label()`

Related Topics

[\\$add_labels_to_models\(\)](#)

[\\$report_models_with_all_labels\(\)](#)

[\\$delete_labels_from_models\(\)](#)

[\\$show_labels\(\)](#)

[\\$hide_labels\(\)](#)

\$report_model_info()

Creates a report window containing information displayed in the Registered Model Info pane of the Component window.

Usage

\$report_model_info()

Component window > **Registered Model Info** pane popup menu > **Report** > **List Area**

Arguments

None.

Return Values

VOID.

Description

The report window contains its own popup menu with items for exporting the information to a printer or file.

Examples

\$report_model_info()

Related Topics

[\\$report_model_entry_info\(\)](#)

[\\$report_models_with_all_labels\(\)](#)

[\\$report_models_for_each_label\(\)](#)

\$report_models_with_all_labels()

Prerequisite: To use this function, you must be in an active Component window.

Produces a report window containing information on the models that have all the specified labels.

Usage

`$report_models_with_all_labels([labels], [part_interfaces])`

Component window > **Registered Model Info** pane popup menu > **Report** > **Models With All Labels**

Arguments

- **labels**

A vector containing one or more strings, each of which is a label name.

- **part interfaces**

A vector of vectors containing information about one or more part interfaces. Each sub-vector contains two strings describing a component and part interface. The format of the sub-vector is:

`["component_pathname", "part_interface"]`

The argument is of the following format:

```
$get_models_with_all_labels(["label_1", "label_2", ..., "label_n"],  
  [[component_path_1", "part_interface_1"], ...["component_path_n",  
    "part_interface_n"]])
```

Return Values

VOID.

Description

Only models from the specified part interface are considered.

Examples

`$report_models_with_all_labels()`

Related Topics

[\\$add_labels_to_models\(\)](#)

[\\$report_models_for_each_label\(\)](#)

[\\$delete_labels_from_models\(\)](#)

[\\$show_labels\(\)](#)

[\\$hide_labels\(\)](#)

\$report_pin_info()

Prerequisite: To use this function, you must be in an active Component window and the Pins pane must be visible.

Creates a report window containing information displayed in the Pins pane of the Component window.

Usage

\$report_pin_info()

Component window > **Pins** pane popup menu > **Report List Area**

Arguments

None.

Return Values

VOID.

Description

The report window contains its own popup menu with items for exporting the information to a printer or file.

Examples

\$report_pin_info()

Related Topics

[\\$hide_pins\(\)](#)

[\\$show_pins\(\)](#)

\$save_components_edits()

Prerequisite: To use this function, you must be in an active Component window.

Saves the edits made to the specified components.

Usage

`$save_components_edits([components])`

Component window > **Component Information** pane popup menu > **Edit** >
Save Selected Component Edits

Arguments

- **components**

A vector of strings, each of which is the name of a component. The vector format is:

```
[“component_pathname_1”, “component_pathname_2”, ... ,  
 “component_pathname_n”]
```

Return Values

VOID.

Description

Component data on disk is updated to include the edits, and then the component's status is returned to the unchanged state.

Components that have undergone changes are indicated by their icon, which is shown on a white background. A component is in the changed state if it has had changes and not been saved.

Examples

```
$save_components_edits(["/user/in_process/741s74a",  
                        "/users/in_process/newcard_reader2 /card_reader"])
```

Related Topics

[\\$add_components\(\)](#)

[\\$remove_components\(\)](#)

[\\$forget_components_edits\(\)](#)

[\\$report_component_info\(\)](#)

\$select_model_object()

Prerequisite: To use this function, you must be in an active Component window and the Registered Model Info pane must be visible.

Selects a model that is currently shown in the Registered Model Info pane of the Component window.

Usage

`$select_model_object("type", mode, object_info)`

Component window > **Registered Model Info** pane popup menu > **Select** > **Select Models By Type**

Arguments

- **type**
A string that specifies the object type of the model. The string can contain wildcards. A label's type is "label".
- **mode**
A name that indicates the select mode to use.
 - **@add** — Adds items to the selected set.
 - **@reselect** — Everything is first unselected and then the specified items are selected. Default.
- **object_info**
A rest argument that is made up of three strings that describe the model. The rest argument format varies with the type of object being selected.
When the type is a model type, the rest argument has the format:
["component pathname", "part interface", "model"]
When the object type is "label", then the rest argument is made up of five strings, as follows:
["component pathname", "part interface", "model", "model type", "label name"]

Return Values

VOID.

Description

The model selected can be added to items already selected, or the selected set can first be cleared so that only the specified model remains selected.

The GUI has defined a single mouse click to select the item under the cursor so that clicking on a model in the Registered Model Info pane selects that model.

This builtin has no effect if the list of models is hidden. The models can be shown using the `$show_model()` function.

Examples

The following command selects a model named \$TRAINING/741s74a/MG_STD, and adds it to the selected set:

```
$select_model_object("mgc_standard", @add, "$PAD/741s74a",  
"MG_STD", "TRAINING/741s74a/MG_STD")
```

The following command selects a label:

```
$select_model_object("label", @reselect, "$PAD/741s74a",  
"MG_STD", "TRAINING/741s74a/MG_STD", "mgc_symbol", "TYPE")
```

Related Topics

[\\$show_model\(\)](#)

[\\$unselect_model_object\(\)](#)

\$select_object()

Prerequisite: To use this function, you must be in an active Component window.

Selects an object (or objects) in the Component Information pane of the Component window based on the specified object information.

Usage

\$select_object()

Component window > **Component Information** pane popup menu > **Select**

Arguments

- **type**
A string that specifies an object type (for example, “mgc_component” or “pi”). The string may be wildcarded, as in the following examples:
 - “*” — To represent objects of any type.
 - “mgc*” — To represent all objects whose type name begins with “mgc”.
- **mode**
A name that indicates the select mode to use.
 - **@add** — Adds items to the selected set.
 - **@reselect** — Everything is first unselected and then the specified items are selected. Default.
- **object_info**
A rest argument that further describes the objects to select. If the type specified is “pi”, the rest argument is two strings describing a part interface and is formatted as follows:
[“component_pathname”, “part_interface_name”]
If the type argument is anything other than “pi”, then the rest argument is a single string which is the object's name, as in “object_pathname”
In either case, whether the rest argument is one or two strings, the object_info argument can also be wildcarded.
Note that all objects including part_interfaces cannot be selected with one invocation or the command even with the use of wildcards. This is because the rest argument is either one or two strings. If two strings, then it is taken as defining a part interface object. If one string, the rest argument is taken as defining something other than a part interface.
Combinations of arguments, along with their results, are as follows:
 - (“mgc_component”, “*”) — To represent any object that is of type “mgc_component”.

- (“*”, “*card*”) — To represent any object other than part interfaces, whose name contains the string “card”.
- (“*”, “*”) — To represent any object of type other than “pi”.
- (“*”, “*”, “*”) — To represent all part interface objects.
- (“*”, “*”, “*micro*”) — To represent any part interface with a name starting with “micro” from any component.

Return Values

VOID.

Description

The object may be added to those already selected, or the selected set may first be cleared so that only the objects specified by the argument remain selected.

Examples

```
function $select_all(), sealed
{
    $select_object("*", @add, "*");
    $select_object("*", @add, "*");
}
```

Related Topics

[\\$unselect_object\(\)](#)

\$set_bgd_color()

Prerequisite: To use this function, you must be in an active Component window.

Sets the color to use for list backgrounds in the Component window.

Usage

```
$set_bgd_color("color")
```

Component window > **Component Information** pane popup menu > **Setup** > **Colors** > **List Background**

Arguments

- **color**
A string that specifies a color name.

Return Values

VOID.

Examples

```
$set_bgd_color("White")
```

Related Topics

[\\$set_fgd_color\(\)](#)

\$set_bgd_color_title_items()

Prerequisite: To use this function, you must be in an active Component window.

Sets the color to use for the background of the label bars in the various Component window panes.

Usage

```
$set_bgd_color_title_items("color")
```

Component window > **Component Information** pane popup menu > **Setup** > **Colors** > **Separator Item Background**

Arguments

- **color**
A string that specifies a color name.

Return Values

VOID.

Examples

```
$set_bgd_color_title_items("Yellow")
```

Related Topics

[\\$set_fgd_color_title_items\(\)](#)

\$set_bgd_color_titles()

Prerequisite: To use this function, you must be in an active Component window.

Sets the color to use for the backgrounds of the list title bars in the various Component window panes.

Usage

```
$set_bgd_color_titles("color")
```

Component window > **Component Information** pane popup menu > **Setup** > **Colors** > **List Title Background**

Arguments

- **color**
A string that specifies a color name.

Return Values

VOID.

Examples

```
$set_bgd_color_titles("Green")
```

Related Topics

[\\$set_fgd_color_titles\(\)](#)

\$set_constraints()

Sets the width and orientation of the Component Information pane of the Component window.

Usage

\$set_constraints()

Component window > **Component Information** pane popup menu > **Setup** > **List Layout**

Arguments

None.

Return Values

- **width_percentage**

A number from 0 to 100 that is taken to be the percentage width of the current window that should be occupied by the Component Information pane when one or more of the other lists are present.

- **orientation** — A name that is either @vertical or @horizontal.
 - **@vertical** — The Registered Model Info, Body Properties, and Pins panes are oriented vertically (column format).
 - **@horizontal** — The additional panes are oriented horizontally (top to bottom).

Description

The percentage controls the width of the Component Information List when other lists are present. If no other lists are present, then the Component Information List expands to fill the window. When other lists are present, the Component Information List occupies a specified percentage of the window size, and the other list takes up the rest of the room. The other lists can be oriented vertically or horizontally. The orientation is specified by the second argument.

\$set_default_part_interface()

Prerequisite: To use this function, you must be in an active Component window.

Sets the default part interface for a specified component to the specified part interface.

Usage

`$set_default_part_interface([part_interface])`

Component window > **Component Information** pane popup menu > **Edit** >
Set Default Part Interface

Arguments

- **part_interface**

A vector of one or more sub-vectors. Each sub-vector describes a part interface to be deleted. The sub-vector contains two string elements in the format:

[“component_pathname”, “part_interface_name”]

The component item can be wildcarded.

Return Values

VOID.

Examples

```
$set_default_part_interface(["/users/in_process/741s74a", "ANSI"]);
```

Related Topics

[\\$delete_part_interfaces\(\)](#)

[\\$set_part_interface_font\(\)](#)

[\\$rename_part_interface\(\)](#)

\$set_fgd_color()

Prerequisite: To use this function, you must be in an active Component window.

Sets the color to use for the list foreground (list text) in the Component window.

Usage

```
$set_fgd_color("color")
```

Component window > **Component Information** pane popup menu > **Setup** > **Colors** > **List Text**

Arguments

- **color**
A string that specifies a color name.

Return Values

VOID.

Examples

```
$set_fgd_color("Black")
```

Related Topics

[\\$set_bgd_color\(\)](#)

\$set_fgd_color_title_items()

Prerequisite: To use this function, you must be in an active Component window.

Sets the color to use for the foreground of the list title bars in the various Component window panes.

Usage

`$set_fgd_color_title_items("color")`

Component window > **Component Information** pane popup menu > **Setup** > **Colors** > **Separator Item Text**

Arguments

- **color**
A string that specifies a color name.

Return Values

VOID.

Examples

`$set_fgd_color_title_items("Black")`

Related Topics

[\\$set_bgd_color_title_items\(\)](#)

\$set_fgd_color_titles()

Prerequisite: To use this function, you must be in an active Component window.

Sets the color to use for the foreground of the list title bars in the Component window panes.

Usage

\$set_fgd_color_titles("color")

Component window > **Component Information** pane popup menu > **Setup** > **Colors** >
List Title Text

Arguments

- **color**
A string that specifies a color name.

Return Values

VOID.

Related Topics

[\\$set_bgd_color_titles\(\)](#)

\$set_font()

Prerequisite: To use this function, you must be in an active Component window.

Sets the font that is used to display text, other than part interface names, in the Component Information pane of the Component window.

Usage

```
$set_font("font_name", redraw)
```

Component window > **Component Information** pane popup menu > **Setup** > **List Font**

Arguments

- **font_name**

A string that is a font name. Selecting the new font name changes the font of the text in the Component window to the selected font style.

- **redraw**

A name that specifies whether or not to redraw the screen using the new font. Options include:

- **@redraw** — Redraws the screen using the new font.
- **@no_redraw** — Does not redraw the screen. Default.

Return Values

VOID.

Examples

```
$set_font("helvetica-bold:14", @redraw)
```

Related Topics

[\\$set_part_interface_font\(\)](#)

\$set_part_interface_font()

Prerequisite: To use this function, you must be in an active Component window.

Sets the font that is used to display part interface names in the Component Information List.

Usage

```
$set_part_interface_font("font_name", redraw)
```

Component window > **Component Information** pane popup menu > **Edit** > **Setup** > **Part Interface Font**

Arguments

- **font_name**

A string that is a font name. Selecting the new font name changes the font of the part interface text in the Component window to the selected font style.

- **redraw**

A name that specifies whether or not to redraw the part interface using the new font.

Options include:

- **@redraw** — Redraws the part interface using the new font.
- **@no_redraw** — Does not redraw the part interface. Default.

Return Values

VOID.

Description

Part Interfaces are not contained objects, and are shown in a different font to distinguish them from other objects actually contained by the component.

Examples

```
$set_part_interface_font("helvetica-oblique:14", @redraw)
```

Related Topics

[\\$delete_part_interfaces\(\)](#)

[\\$set_default_part_interface\(\)](#)

[\\$rename_part_interface\(\)](#)

[\\$set_font\(\)](#)

\$show_body_props()

Prerequisite: To use this function, you must be in an active Component window and the Body Properties pane must be hidden.

Shows the Body Properties pane in the Component window.

Usage

`$show_body_props()`

Component window > **Body Properties** pane popup menu > **Show Body Properties List**

Arguments

None.

Return Values

VOID.

Description

Other panes that are visible shrink in size to make room for the Body Properties List.

Examples

`$show_body_props()`

Related Topics

[\\$hide_body_props\(\)](#)

[\\$report_body_prop_info\(\)](#)

\$show_labels()

Prerequisite: To use this function, you must be in an active Component window and the labels must be hidden in the Registered Model Info pane.

Shows the display of labels in the Registered Model Info pane of the Component window.

Usage

\$show_labels()

Component window > **Registered Model Info** pane popup menu > **Show Labels**

Arguments

None.

Return Values

VOID.

Description

Causes labels to be shown underneath the models in the Registered Model Info pane of the Component window.

Examples

\$show_labels()

Related Topics

[\\$add_labels_to_models\(\)](#)

[\\$report_models_for_each_label\(\)](#)

[\\$delete_labels_from_models\(\)](#)

[\\$report_models_with_all_labels\(\)](#)

[\\$hide_labels\(\)](#)

\$show_model()

Prerequisite: To use this function, you must be in an active Component window and the Registered Model Info pane must be hidden.

Shows the Registered Model Info pane in the Component window.

Usage

`$show_model()`

Component window > **Registered Model Info** pane popup menu > **Show Models List**

Arguments

None.

Return Values

VOID.

Description

This function causes the Models List to become visible. Other panes that are visible may shrink in size to make room for the Registered Model Info pane.

Examples

`$show_model()`

Related Topics

[\\$hide_model\(\)](#)

\$show_pin_properties()

Prerequisite: To use this function, you must be in an active Component window and the pin properties in the Pins pane must be hidden.

Shows the pin properties that may be hidden in the Pins pane of the Component window.

Usage

\$show_pin_properties()

Component window > **Pins** pane popup menu > **Show Pin Properties**

Arguments

None.

Return Values

VOID.

Description

This function causes the pin properties to become visible underneath the pins in the Pins pane of the Component window. Other panes that are visible may shrink in size to make room for the Pins pane.

Examples

\$show_pin_properties()

Related Topics

[\\$hide_pin_properties\(\)](#)

\$show_pins()

Prerequisite: To use this function, you must be in an active Component window and the Pins pane must be hidden.

Shows the Pins pane of the Component window.

Usage

\$show_pins()

Component window > **Component Information** pane popup menu > **Show Pins List**

Arguments

None.

Return Values

VOID.

Description

Other panes that are visible may shrink in size to make room for the Pins pane.

Examples

\$show_pins()

Related Topics

[\\$hide_pins\(\)](#)

[\\$report_pin_info\(\)](#)

\$unselect_model_object()

Prerequisite: To use this function, you must be in an active Component window with the Registered Model Info pane visible, and at least one model must be selected in the pane.

Unselects an object that is currently selected in the Registered Model Info pane of the Component window.

Usage

```
$unselect_model_object("type", object_info)
```

Component window > **Registered Model Info** pane popup menu > **Select** > **Unselect All Models**

Arguments

- **type**
A string that specifies the object type of the model. The string can contain wildcards. A label's type is "label".
- **object_info**
A rest argument that is made up of three strings that describe the model. The rest argument format varies with the type of object being selected. When the type is model type, the rest argument has the format:

["component pathname" "part interface", "model"]

when the object type is label then the rest argument is made up of five strings:

["component pathname" "part interface", "model" "model type", "label name"]

Return Values

VOID.

Description

The GUI has defined a single mouse click to select the item under the cursor so that clicking on a model in the Registered Model Info pane selects that model.

This builtin has no effect if the list of models is hidden. The models can be shown using \$show_model().

Examples

```
$unselect_model_object("mgc_symbol", $PAD/741s74a", "MG_STD",  
"$TRAINING/741s74a/MG_ST D");
```

Related Topics

[\\$select_model_object\(\)](#)

[\\$show_model\(\)](#)

\$unselect_object()

Prerequisite: To use this function, you must be in an active Component window.

Unselects an object in the Component Information pane of the Component window based on the specified object information.

Usage

\$unselect_object()

Component window > **Component Information** pane popup menu > **Select > Unselect All**

Arguments

- **type**

A string that specifies an object type. The string may be wildcarded, as in the following examples:

- **"*"** — To represent objects of any type.
- **"mgc*"** — To represent all objects whose type name begins with "mgc".

- **object_info**

A rest argument that further describes the objects to select. If the type specified is "pi", the rest argument is two strings describing a part interface and is formatted as follows:

[**"component_pathname"**, **"part_interface_name"**]

If the type argument is anything other than "pi", then the rest argument is a single string which is the object's name, as in **"object_pathname"**.

In either case where the rest argument is one or two strings, the **object_info** argument can be wildcarded.

Note that all objects including part_interfaces cannot be unselected with one invocation of the command even with the use of wildcards. This is because the rest argument is either one or two strings. If two strings, then it is taken as defining a part interface object. If one string, the rest argument is taken as defining something other than a part interface.

Combinations of arguments, along with their results, are as follows:

- (**"mgc_component"**, **"*"**) — To represent any object that is of type "mgc_component".
- (**"*"**, **"*card*"**) — To represent any object other than part interfaces, whose name contains the string "card".
- (**"*"**, **"*"**) — To represent any object of type other than "pi".
- (**"*"**, **"*"**, **"*"**) — To represent all part interface objects.
- (**"*"**, **"*"**, **"micro*"**) — To represent any part interface with a name starting with "micro" from any component.

Return Values

VOID.

Examples

```
function $select_all(), sealed
{
    $unselect_object("", @add, "");
    $unselect_object("", @add, "");
}
```

Related Topics

[\\$select_object\(\)](#)

\$validate_models()

Prerequisite: To use this function, you must be in the Registered Model Info pane of the Component window and must select a model in the list.

Performs a validity check on the specified models.

Usage

`$validate_models([models], report_window)`

Component window > **Registered Model Info** pane popup menu > **Edit** > **Validate Models**

Arguments

- **models**

A vector which describes models. the vector is actually a vector of 1 or more sub-vectors. Each sub-vector describes one model in the format:

[component_pathname, part_interface, model_pathname, model_type]

- **report_window**

A switch that can be set to either @no_report or @report.

- **@report** — Causes a report window to be created. Default.
- **@no_report** — Information regarding the validity of models appears as messages in transcript.

Return Values

VOID.

Description

A check is first made to insure the model exists and then the validity of the model is checked. Information on the validation process can display in a report window or the transcript.

Note that the existence of a model may change as the location map changes, the model is moved, and so on. It is possible for the \$validate_models() function to produce error messages due to connection problems, even though the model itself may be valid.

Examples

```
$validate_models(["/user/in_process/741s74a", "MG_STD",  
"/users/in_process/741s74a/MG_STD", "mgc_symbol"], @no_report)
```

Related Topics

[\\$register_models\(\)](#)

[\\$report_model_entry_info\(\)](#)

Chapter 7

Shell Command Dictionary

The following topics in this chapter describe the command reference information for Pyxis Project Manager shell scripts and DDMS shell scripts. Mentor Graphics shell scripts are normally found in *\$MGC_HOME/bin*.

Common Shell Command Usage	724
Shell Command Summary	730

Common Shell Command Usage

The syntax of the design management shell commands corresponds roughly to the syntax of standard UNIX commands. In addition, some usage practices apply to all of the shell commands.

The following sections describe the usage practices that all design management shell commands share in common:

Finding the Name and Type of a Design Object in the Shell	724
Switches, Labels, and Names	725
About Supplying Arguments on Standard Input.	726
About Supplying Options to Design Management Shell Commands.	727
Object Specifiers.	728
Error Handling	729

Finding the Name and Type of a Design Object in the Shell

In general, to specify a design object, you must know its name and its type.

Procedure

1. Change your working directory to the location of the design object whose name and type you need using **/bin/cd**.
2. Run this command:

```
$ list_contents .
```

The design objects that reside in the current working directory display, one per line, in the following format:

```
./my_object:my_type
```

where *my_object* is the name of the design object, and *my_type* is its type.

The following list describes two problems you might encounter when using this step:

- The **list_contents** command cannot recognize your design object.

This happens if the type registry that contains your type is not loaded into your environment. In this case, **list_contents** simply lists the individual members of your design object's fileset as design objects. These files look like this:

```
./my_object:Mgc_container  
./my_object.my_type.attr:Mgc_file  
./my_object.suffix1:Mgc_file
```

./my_object.suffix2:Mgc_file

...

The filesset of *my_object* consists of a directory and several files, each with a unique suffix. The file with the “.attr” suffix is the attribute file for the design object, and the object's type, *my_type*, is embedded in its name.

If the individual filesset members of your design object are listed as in the preceding example, then refer to Troubleshooting section listed in the Related Topics.

- The operating system cannot find `list_contents` or any other design management shell command. Note that all design management commands reside in *\$MGC_HOME/bin*.

If this occurs, then see your System Administrator.

Related Topics

[Troubleshooting](#)

[list_contents](#)

Switches, Labels, and Names

The design management shell commands feature three kinds of arguments—switches, labels, and names.

Switches and labels begin with a hyphen; names do not. Switches and labels set the options that govern the action of the command as a whole; names provide operands upon which the command works.

The following rules apply to the use of switches, labels, and names:

- Follow a label immediately with a value argument.
-v 5 # correct
- Do not follow a switch with a value argument.
-r 5 # syntax error
- Provide each switch of the design management shell commands with its own hyphen. (This notation differs from standard UNIX usage.)
- Precede any name arguments with the switches and labels on the command line. Switches and labels may be intermixed among themselves in any order.

```
set_version_depth -d 5 my_obj:my_type -s # wrong
```

```
set_version_depth -d 5 -s my_obj:my_type # correct
```

```
set_version_depth -s -d 5 my_obj:my_type # correct
```

- If a command requires more than one name argument, you must supply some multiple of the required number of name arguments. The command treats leftover arguments as an error. The following example is in error because the **move_object** command requires two name arguments:

```
move_object -r from_obj:its_type # error
```

Related Topics

[Common Shell Command Usage](#)

About Supplying Arguments on Standard Input

At the command line, you can direct data to the standard UNIX input stream. You can use exactly one of the following three sources for standard input at a time — a file, the pipe operation, or the keyboard.

1. A file:

In the following example, the **set_version_depth** command gets its arguments from the file *arg_file*.

```
set_version_depth -d 5 < arg_file
```

2. A pipe, which redirects the preceding command's output:

In the following example, the **set_version_depth** command gets its arguments from the output of the **list_contents** command.

```
list_contents my_dir | set_version_depth -d 5
```

If you try to use both a file and a pipe at the same time, UNIX ignores the pipe and uses the file.

3. The keyboard, in the absence of a file or pipe:

In the following example, you enter arguments for the **set_version_depth** command directly at the keyboard. However, instead of placing them on the command line, you enter them as a list. You end the list with a Ctrl-Z, the UNIX end-of-file signal.

```
set_version_depth -d 5

my_name:my_type
another_name
<EOF>
```

Related Topics

[Common Shell Command Usage](#)

About Supplying Options to Design Management Shell Commands

Each design management shell command provides the dash (-) option, which mimics the UNIX usage. The dash option directs a command to accept name arguments from the standard input. This option looks like a switch, but it does not follow the syntax of a switch. Instead, it follows all other arguments on the command line.

You use the dash option only when you want to supply name arguments from both the command line and from standard input. The command looks automatically for name arguments on standard input if there are none on the command line. In that case, using the dash option is acceptable but superfluous. The following two commands are equivalent:

```
list_contents my_dir | set_version_depth -d 5 -
```

```
list_contents my_dir | set_version_depth -d 5
```

The following rules apply to the use of the dash option:

- You put all switches and labels on the command line. The commands treat all input from the standard input stream, including switches and labels, as arguments.
- Required name arguments need not appear on the command line if the command can find them on standard input.
- You can supply name arguments both on the command line and from standard input. In this case, the command processes the arguments from the command line first.
- If a command requires more than one argument, and you supply arguments both on the command line and from standard input, they are treated as two separate sets of arguments, and each must contain a whole multiple of the required number. The command treats left over arguments as an error. The following example is in error because the **move_object** command requires two name arguments:

```
move_object -r from_obj:its_type - < input_file # error
```

Related Topics

[Common Shell Command Usage](#)

Object Specifiers

An object specifier is a command argument that you supply to identify the design object on which you want a command to work.

An object specifier has three parts:

1. **Name specifier** — The pathname of the design object, required in all object specifiers
2. **Type specifier** — The type of the design object, optional in all object specifiers
3. **Version specifier** — The number of one version of the design object, optional in object specifiers for some commands, and not permitted in others

The three parts of an object specifier all belong to one string without spaces. A colon precedes a type specifier if it is present. Square brackets surround a version specifier, if it is present. Because of the brackets, if you use the version specifier in a shell other than the Bourne shell, you put the entire object specifier in quotes. The following object specifier indicates version 3 of the design object named *my_obj*, of type *my_type*:

my_obj:my_type[3]

When the name specifier is specified as a relative pathname, it is resolved using the value of the MGC_WD environment variable, if it is valid. If the value of the MGC_WD environment variable is not valid, the current working directory is used to resolve the relative pathname. In either case, the relative pathname is resolved by appending the value of MGC_WD or the current working directory to the beginning of the relative pathname.

Several design objects of different types might have the same name. The type specifier indicates the design object of a specific type. The following example shows the three correct forms of syntax that include the type specifier:

my_obj:my_type[3]

my_obj:my_type[]

my_obj:my_type

If you omit the type specifier, the command works on all design objects of the given name, no matter what their type. The following example shows three correct forms of syntax that omit the type specifier:

my_obj[3]

my_obj[]

my_obj

Some commands work on objects as a whole; with such commands, you must not supply a version specifier. Other commands work on a single version of a design object; with such commands, the version specifier is optional. When permissible, you use the version specifier to indicate the version number on which you want to work. The following example shows two correct forms of syntax that explicitly indicate the version number:

my_obj:my_type[3]

my_obj[3]

If you omit the version specifier or if you omit just the version number, a command that works on a version uses the current version of the design object. The following example shows four correct forms of syntax that explicitly or implicitly indicate the current version:

my_obj:my_type[]

my_obj:my_type

my_obj[]

my_obj

Related Topics

[Common Shell Command Usage](#)

Error Handling

When a command encounters a syntax error, it performs no work on the design object or objects that you give it as arguments but simply returns the error to you.

When a command returns an error, it performs two actions:

- Displays an error message
- Sets an error condition in the form of a non-zero return value. UNIX recognizes and uses this value to terminate processing of a series of commands

When a command encounters no syntax errors, it processes as many objects as it can from the entire argument list. If one or more of these arguments causes a usage error, the error has no effect upon the processing of other sets of arguments. The command displays an error message for each usage error it encounters. It returns an error condition if one or more usage errors occur.

You can test for the error condition in a shell script, for the script to take an appropriate action. Although you can use an option to suppress the error message, the error condition remains intact.

Related Topics

[Common Shell Command Usage](#)

Shell Command Summary

The table in this section describes the design management shell commands available in the Pyxis Project Manager application.

Table 7-1. Shell Commands

Function	Description
change_ncf_ref	Updates references to the specified component, symbol, or library in a related Component NCF.
chref	Changes the references in the objects contained within the specified directories.
change_references	Changes the reference pathnames and reference version numbers of a design object.
checkref	Checks the references of all objects found in the specified directories to determine if the version of the object pointed to by each reference exists.
copy_object	Copies a design object to a new location.
copy_version	Copies a single version of a design object to a new location.
delete_object	Deletes a design object.
ddms_locenv	Reads a location map and defines shell environment variables based on the entries in the map.
ddms_which_map	Goes through the search rules and returns the full pathname of the location map that would be found and used by an application.
dmgr_ic	Invokes the Pyxis Project Manager application.
freeze_version	Freezes a version of a design object, preventing it from being deleted due to version depth.
get_hard_name	Converts a list of pathnames to hard pathnames.
get_soft_name	Converts a list of pathnames to soft pathnames.

Table 7-1. Shell Commands (cont.)

Function	Description
list_contents	Lists the design objects in a directory or list of directories.
listref	Lists the references of all objects found in the specified directories and their sub-directories.
list_references	Lists the references of a design object.
move_object	Moves a design object to a new location.
revert_version	Deletes the current version of a design object and makes the previous version current.
salvage_object	Salvages a damaged design object caused by an application failure.
set_version_depth	Specifies how many versions a design object keeps.
show_object_info	Displays information for the specified design object or objects.
unfreeze_version	Unfreezes a frozen version of a design object.

change_ncf_ref

Updates pathnames within the related Component NCF when a specified directory, component, or symbol has been copied or moved/renamed.

Usage

```
change_ncf_ref [src_path dest_path]  
    [-c src_component_path target_component_path]  
    [-sc src_component_path src_symbol_name target_symbol_name]  
    [-sm src_component_path src_symbol_name target_symbol_name]
```

Arguments

- **-c**
Updates the NCF that resides in the target component when a component is moved/renamed or copied; Replaces each occurrence of the path, symbol, or component specified by *src_component_path* with that specified by *target_component_path*.
- **-sc**
Updates the NCF that resides in the *src_component_path* when a symbol is copied within the *src_component_path*; Replaces each occurrence of the *src_symbol_name* with the *target_symbol_name*.
- **-sm**
Updates the NCF that resides in the *src_component_path* when a symbol is moved/renamed within the *src_component_path*; Replaces each occurrence of the *src_symbol_name* with the *target_symbol_name*.
- ***src_path***
The pathname to the source directory/library that was copied, moved, or renamed.
- ***dest_path***
The pathname to the target directory/library that was copied, moved, or renamed.
- ***src_component_path***
The pathname to the source component that was copied, moved, or renamed.
- ***target_component_path***
The pathname to the target component that was copied, moved, or renamed.
- ***src_symbol_name***
The name of the source symbol that was copied moved or renamed.
- ***target_symbol_name***
The name of the target symbol that copied moved or renamed.

Return Values

None.

Description

If no arguments are supplied, the command usage information displays.

For more information on Component-type NCFs, see the [Pyxis Netlister User's and Reference Manual](#).

Examples

Use the following syntax to update a component NCF after a library is moved:

```
change_ncf_ref <src_path> <dest_path>
```

change_references

Changes the reference pathnames and reference version numbers of the specified design object(s).

Usage

`change_references` [*options*] pattern replacement {object_specifier}

Arguments

- **pattern**
Identifies the string pattern or patterns to be replaced in the reference pathnames. The pattern can include only the pathnames of design objects, not their types or versions. The pattern is a UNIX regular expression.
- **replacement**
Defines the string to replace any string that matches the specified pattern.
- **object_specifier**
Identifies a list of design objects and version numbers whose references are to be changed. If the argument includes a type specifier, the list contains just the one specified design object. If the argument has no type specifier, the list contains all the types of design objects that have the given name. If **change_references** cannot change the references in the specified versions of all the design objects on the list, it changes as many as it can, but returns an error.
- **-a**
All versions—applies changes to the references in all versions of the design object or objects. Each version of a design object might hold its own set of references to other objects. This option applies the specified changes to the references held by each version. When you supply this option, **change_references** ignores the version specifiers in all object specifier arguments. This option is not related to the **-v** (version) option. As a result, you can specify both options.
- **-h**
Help—displays online help and exits the command.
- **-s**
Silent—suppresses the display of all messages, including error messages. This option does not affect the error condition.
- **-v version_number**
Version—in references that match the pattern, sets the version number to the specified version number. This option does not relate to versions of the design object arguments, but to the version number on references. As a result of changing this number, the reference cites a different version of the referenced design object. Also, this option does not replace patterns that relate only to the pathnames on references. However, this option changes the

version numbers on only those references whose pathname matches the pattern argument. This option is not related to the **-a** (all versions) option. As a result, you can specify both options.

Return Values

VOID.

Description

This shell command searches the specified reference pathnames for the first matching string that match the pattern argument. Because this pattern can be a UNIX regular expression, it can match many different pathname patterns. However, it can only match the first successful match in each reference. To replace multiple strings in a single reference, you must execute this shell command multiple times. When **change_references** finds a matching string, it replaces the matched string with the specified replacement string. In addition, if you supply the **-v** option and a version number, **change_references** changes the version number on the reference to the number supplied. The result of this change is that **change_references** changes which version of the referenced design object that the reference points to.

The **change_references** command does not return an error when it finds no strings that match the pattern.

By default, **change_references** works on single versions of each specified design object. As a result, you can include version specifiers in the object specifier arguments.

Examples

1. This example changes the references held in the current version of the “my_obj” design object of “my_type” type. In those references, the **change_references** command changes the first occurrence of the pattern “old” to the string “new”.

```
change_references old new my_obj:my_type
```

If the current version of my_obj had references to *\$PROJ_XYZ/proj/old/doc* and *\$PROJ_XYZ/proj/old/picture*, then this command would change those references to *\$PROJ_XYZ/proj/new/doc* and *\$PROJ_XYZ/proj/new/picture*.

2. This example changes the references held in the current version of the “my_obj” design object of “my_type” type. In those references whose pathnames contain the string “old”, the version of the object pointed to by the reference changes to 2.

```
change_references -v 2 old old my_obj:my_type
```

Because the pattern and the replacement string are the same, the **change_references** command changes no pathnames in this example. Therefore, if the current version of my_obj references version 4 of *\$PROJ_XYZ/proj/old/doc* and references version 3 of *\$PROJ_XYZ/proj/old/picture*, this command would change those references to version 2 of each design object.

3. This example changes the references held in all versions of the design object named “my_obj” of “my_type” type. In each of those references, the change_references command changes the first occurrence of one of the “bad”, “bed”, and “bid” strings to the “bud” string.

change_references -a b[aei]d bud my_obj:my_type

4. This example changes the references held in version 7 of any type of design object named “my_obj”. In those references, the change_references command changes the first occurrence of one of the “u”, “uum”, “umm” strings, and so on, to the “oops” string.

change_references um* oops my_obj[7]

Related Topics

[chref](#)

[copy_object](#)

[\\$copy_version\(\)](#)

[list_contents](#)

[list_references](#)

[move_object](#)

checkref

Checks the references of all the design objects found in the specified directories to determine whether the version of the object pointed to by each reference exists.

Usage

checkref [*options*] {pathnames}

Arguments

- **pathnames**
Identifies the directories that contain the objects whose references are checked. Each directory can be specified as either an absolute or a relative pathname.
- **-h**
Help—displays online help.

Return Values

VOID.

Description

If any referenced version does not exist, the command displays an error message.

If you specify the pathnames argument as a relative pathname, it is resolved using the value of the MGC_WD environment variable, if it valid; a message displays the relative pathname and the new resolved pathname. If the value of MGC_WD is not valid, the current working directory is used to resolve the relative pathname.

Examples

This example checks the references held in all versions of the “cell” design object and reports the errors as shown in the sample output below.

checkref cell

The following lines show a sample output for this shell command:

```
Object /net/berg/mult/gen_lib./part (Eddm_part) not accessible
Object /net/berg/mult/gr.test.mult_mfa/as_lib.4as04/part
(Eddm_part) not accessible
Object /net/berg/mult/gr.test.mult_mfa/as_lib.4as08/part
(Eddm_part) not accessible
```

Related Topics

[chref](#)

[list_references](#)

[change_references](#)

chref

Changes the references in all versions of the objects contained within the specified directories.

Usage

`chref [options] pattern replacement {pathname}`

Arguments

- **pattern**
Identifies the string that is the pathname pattern within the references to be replaced. The pattern can include only the pathnames of design objects, not their types or versions. The pattern argument is a UNIX regular expression.
- **replacement**
Defines the string that is substituted for all reference pathname text which matches the pattern argument.
- **pathname**
Identifies the directories that contain the objects whose references are changed. Each directory can be specified as either an absolute or a relative pathname. The pathname argument can be specified using wildcards.
- **-h**
Help—displays online help and exits the command.

Return Values

VOID.

Description

The `chref` command changes the pathnames of references in all versions of the specified design objects. If you specify the pathname argument as a relative pathname, it is resolved using the value of the `MGC_WD` environment variable, if it is valid; a message displays the relative pathname and the new resolved pathname. If the value of `MGC_WD` is not valid, the current working directory is used to resolve the relative pathname.

This command searches the specified reference pathnames for strings that match the pattern argument. Because this pattern can be a UNIX regular expression, it can match many different pathname patterns. When the `chref` command finds a matching string, it replaces the matched string with the specified replacement string.

The `chref` command does not return an error when it finds no strings that match the pattern.

Examples

The following example changes the references held in all versions of the objects found in the `$PROJ_XYZ/design` directory. In those references, the `chref` command changes all occurrences of the “cell” pattern to the “part” string.

chref cell part \$PROJ_XYZ/design/config_object

Related Topics

[change_references](#)

[list_references](#)

copy_object

Copies a design object to a target location.

Usage

`copy_object [options] {src_object_specifier dest_object_specifier}`

Arguments

- **src_object_specifier**

Identifies a list of design objects to be copied. If the argument includes a type specifier, the source list contains just the one specified design object. If the argument has no type specifier, the source list contains all the types of design objects that have the given name. If the `copy_object` command cannot copy all the design objects on the source list, it copies as many as it can, but returns an error.

- **dest_object_specifier**

Identifies a list of design objects to hold the copies of the source list. If the argument includes a type specifier, the destination list contains just the one specified design object. If the argument has no type specifier, the destination list contains all the types of the design objects that are on the source list. For each type of design object on the source list, the destination list must contain an identical type of design object, or else the `copy_object` command returns an error.

If a source design object already exists at the destination, the `copy_object` command returns an error, unless the replacement switch **-r** is specified.

- **-h**

Help—displays online help and exits the command.

- **-r**

Replace—replaces an existing destination design object with a copy of the specified source design object. If no design object already exists at the destination, the `copy_object` command ignores this option and creates a new design object.

- **-s**

Silent—suppresses the display of all messages, including error messages. This option does not affect the error condition.

Return Values

VOID.

Description

The `copy_object` command copies a design object, including all of its versions, from the specified source location to the specified destination. To copy just one version of a design object, you use the **copy_version** command.

If the source design object is a directory or a container (that is, it has a directory as a fileset member), the `copy_object` command descends into the container and copies each object in the directory. Although some versions of the source design object might be frozen, the `copy_object` command leaves all destination design object versions unfrozen.

If no design object exists at the destination, the `copy_object` command creates a new one. If a design object already exists at the destination, you use the **-r** (replacement) option; otherwise, the `copy_object` command returns an error.

The references of the destination design object are the same as those of the source design object. That is, they might still point to the source design object or its contents. To repair these or any other references to reflect the new location, you use the `change_references` command.

Because the `copy_object` command copies each specified design object as a whole, you cannot use version specifiers in the *src_object_specifier* and *dest_object_specifier* arguments. If you do, the `copy_object` command returns an error.

Caution

If you omit the type specifier portion of *object_specifier*, `copy_object` attempts to copy every design object of any type whose name matches the name specifier that you provide. This may result in some non-design object files or directories (file system objects that are not part of a recognized design object) with the same name being copied, along with the design objects you intended to copy. To avoid inadvertently copying files and directories, you should always provide the type specifier along with the name specifier to the `copy_object` shell command.

Examples

This example copies the design object named “my_obj” of “my_type” type in the current working directory to the “his_obj” design object of my_type type, also in the current working directory.

```
copy_object -r my_obj:my_type his_obj
```

If the current working directory already contains a design object his_obj of my_type type, the previous command replaces it with a copy of my_obj.

Related Topics

[change_references](#)

[move_object](#)

[\\$copy_version\(\)](#)

[unfreeze_version](#)

[freeze_version](#)

copy_version

Copies a single version of a design object to a new location.

Usage

`copy_version [options] {src_object_specifier dest_object_specifier}`

Arguments

- **src_object_specifier**

Identifies a list of design objects and version numbers that you want to copy. If the argument includes a type specifier, the source list contains just the one specified design object. If the argument has no type specifier, the source list contains all the types of design objects that have the given name. If the `copy_version` command cannot copy the specified versions of all the design objects on the source list, it copies as many as it can, but returns an error.

- **dest_object_specifier**

Identifies a list of design objects to hold the copies of the source list. If the argument includes a type specifier, the destination list contains just the one specified design object. If the argument has no type specifier, the destination list contains all the types of the design objects that are on the source list. For each type of design object on the source list, the destination list must contain an identical type of design object, or else the `copy_version` command returns an error.

If a source design object already exists at the destination, the `copy_version` command returns an error, unless the replacement switch **-r** is specified.

- **-a**

Add—adds the source design object version to the destination design object as a new version, making it the destination's current version. If no design object already exists at the destination, the `copy_version` command ignores this option and creates a new design object with one version.

However, if the source and destination design objects are unversioned, this option replaces the destination design object. In that case, it has the same effect as the **-r** (replacement) option.

If you try to specify both the **-a** and **-r** (replacement) options, the `copy_version` command returns an error.

- **-h**

Help—displays online help and exits the command.

- **-r**

Replace—replaces an existing destination design object with a design object that has one version. That version is a copy of the specified version of the source design object. If no design object already exists at the destination, the `copy_version` command ignores this option and creates a new design object with one version.

If the source and destination design objects are unversioned, then this option replaces the entire destination design object. This replacement is equivalent to issuing the `copy_object` command with the **-r** (replacement) option.

If you try to specify both **-r** and **-a** (add) options, the `copy_version` command returns an error.

- **-s**

Silent—suppresses the display of all messages, including error messages. This option does not affect the error condition.

Return Values

VOID.

Description

The `copy_version` command copies a single version of a design object from the specified source location to the specified destination. To copy a whole design object, including all of its versions, in one operation, you use the `copy_object` command.

If the source design object is unversioned, the `copy_version` command copies the whole object. If the source design object is a directory or a container (that is, it has a directory as a fileset member), the `copy_version` command copies the directory, but not its contents. To copy a directory and specific versions of its contents, you use the `copy_version` command repeatedly, copying each object within the directory in a separate operation. Although the source design object version might be frozen, the `copy_version` command leaves the destination design object version unfrozen.

If no design object exists at the destination, the `copy_version` command creates a new one that has only one version. If a design object does already exist at the destination, you must use either the **-a** (add) or **-r** (replacement) option. Otherwise, the `copy_version` command returns an error.

The references of the destination design object are the same as those of the source design object. That is, they might still point to the source design object or its contents. To repair these or any other references to reflect the new location, you use the `change_references` command.

Because the `copy_version` command copies only one version of each specified design object, you can use a version specifier in the *src_object_specifier*. If you use a version specifier in the *dest_object_specifier* argument, the `copy_version` command returns an error.

Examples

1. This example copies version 3 of the design object at `$PROJ_XYZ/proj/my_obj` of “my_type” type to “his_obj” in the current working directory.

```
copy_version -r $PROJ_XYZ/proj/my_obj:my_type[3] his_obj
```

The new `his_obj` has one version: version number 1. If the current working directory already contains a `his_obj` design object of `my_type` type, the previous command replaces it with version 3 of `my_obj`.

2. This example copies version 3 of the design object at *\$PROJ_XYZ/proj/my_obj* of “my_type” type , and adds it as the current version of “his_obj” in the current working directory.

```
copy_version -a $PROJ_XYZ/proj/my_obj:my_type[3] his_obj
```

Related Topics

[change_references](#)

[copy_object](#)

[freeze_version](#)

[unfreeze_version](#)

ddms_locenv

Reads a location map and defines shell variables based on the entries found in the location map.

Usage

```
ddms_locenv -c[sh] -s -h -i | map_file output_file
```

Arguments

- **-i**
Required if you do not specify the *map_file* argument. Read input from a location map found according to the search rules, rather than from a specified *map_file*. Uses the **ddms_which_map** shell script to follow the search path rules. You can specify only one *map_file*.
- **map_file**
Required if you do not specify the **-i** switch. Name of the file containing the location map.
- **output_file**
Name of the file that contains commands to set environment variables for each entry in the location map that has at least one hard pathname.
- **-c [sh]**
Indicates that the commands produced in the output file should be in C shell format, with the intent that these commands be executed via a C shell **source** command. If this switch is not present, the output is in Bourne shell format.
- **-h**
Help—displays the help message.
- **-s**
Silent flag—suppresses output of the line describing how to execute the output file.

Return Values

VOID.

Description

The shell variables are active in the shell from which the script was called and in its subshells. The variables are external to the calling script.

This script can be executed from any shell. The output of the script is another script that you execute to produce the shell definitions. This output script is produced in either Bourne or C shell format, depending on whether you specify the **-c** switch. A message optionally displays, giving the exact command line the user types to execute the output script.

Examples

1. The following Bourne shell example assumes the user has a location map called “my_map”, which causes the variables shown in the example to be defined.

```
ddms_locenv my_map exec_file  
// In a Bourne Shell, type the following:
```

```
.exec_file  
// Look at variable definitions  
set  
...
```

The following is the output from the script or operating system:

```
MY_FIRST_SOFT_NAME=/hard/name  
MY_SECOND_SOFT_NAME=/hard/name2
```

2. The following is an example for the C shell:

```
ddms_locenv -csh my_map exec_file  
// In a C Shell, type: source exec_file  
source exec_file  
printenv  
...
```

The following is the output from the script or operating system:

```
MY_FIRST_SOFT_NAME=/hard/name  
MY_SECOND_SOFT_NAME=/hard/name2
```

Related Topics

[ddms_which_map](#)

ddms_which_map

Goes through the search rules and returns the full pathname of the location map file that would be found and used by an application.

Usage

```
ddms_which_map -s
```

Arguments

- **-s**

Silent mode—suppressed output if no location map is found.

Return Values

VOID.

Description

If no location map is used, the script displays a message to that effect. If the pathname provided is wrong, “Not found” is printed after the file name.

Examples

The following examples show the use of ddms_which_map:

1. Example 1:

```
setenv MGC_LOCATION_MAP NO_MAP
ddms_which_map -s
ddms_which_map
```

The following is the output from the script or operating system:

```
No location map is used since $MGC_LOCATION_MAP = NO_MAP
```

2. Example 2:

```
setenv MGC_LOCATION_MAP /pad/hello_everybody
ddms_which_map -s
ddms_which_map
```

The following is the output from the script or operating system:

```
/pad/hello_everybody NOT FOUND
```

3. Example 3:

```
setenv MGC_LOCATION_MAP /pad/hello_everybody
echo "some text" > /pad/hello_everybody
ddms_which_map -s
```

The following is the output from the script or operating system:

```
/pad/hello_everybody
```

4. Example 4:

```
setenv MGC_LOCATION_MAP /pad/hello_everybody  
ddms_which_map
```

The following is the output from the script or operating system:

```
/pad/hello_everybody
```

Related Topics

[ddms_locenv](#)

delete_object

Deletes a design object.

Usage

`delete_object` [*options*] {*object_specifier*}

Arguments

- **object_specifier**

Identifies a list of design objects to be deleted. If the argument includes a type specifier, the list contains just the one specified design object. If the argument has no type specifier, the list contains all the types of design objects that have the given name. If the `delete_object` command cannot delete all the design objects on the list, it deletes as many as it can, but returns an error.

- **-h**

Help—displays online help and exits the command.

- **-s**

Silent—suppresses the display of all messages, including error messages. This option does not affect the error condition.

Return Values

VOID.

Description

The `delete_object` command deletes all versions of a design object, including, frozen and unfrozen. If the specified design object is a directory or a container (that is, it has a directory as a fileset member), the `delete_object` command deletes the contents of the directory.

If you try to delete a design object that does not exist, an error is returned.

Because the `delete_object` command deletes each specified design object as a whole, you cannot use version specifiers in the *object_specifier* argument. If you do, then the `delete_object` command returns an error.

Caution



If you omit the type specifier portion of *object_specifier*, `delete_object` attempts to delete every design object of any type whose name matches the name specifier that you provide. This may result in some non-design object files or directories (file system objects that are not part of a recognized design object) with the same name being deleted, along with the design objects you intended to delete. To avoid inadvertently deleting files and directories, you should always provide the type specifier along with the name specifier to the `delete_object` shell command.

Examples

1. This example deletes the design object at *\$PROJ_X/proj/my_obj* of “my_type” type.

```
delete_object "$PROJ_X/proj/my_obj:my_type"
```

2. This example deletes all design objects of “my_type” type in the directory at *\$PROJ_X/proj*.

```
list_contents -t my_type "$PROJ_X/proj" | delete_object
```

Related Topics

[copy_object](#)

[freeze_version](#)

[list_contents](#)

[move_object](#)

[unfreeze_version](#)

dmgr_ic

Invokes the Pyxis Project Manager application in a shell window.

Usage

`dmgr_ic [options]`

Arguments

- **-nodisplay**

NODisplay: Invokes the Pyxis Project Manager application in nodisplay mode. The nodisplay mode lets you use AMPLE-based functions without using the Pyxis Project Manager graphical interface. At the shell level, you can supply a series of functions, one line at a time, or a pathname to an executable script.

- **-nostart**

Nostart: Causes the Pyxis Project Manager application to invoke without executing the startup files. Although none of the startup files, *dmgr_default.startup*, *dmgr.startup*, nor *dmgr_toolbox_path.startup* are executed, you are still able to save the session defaults and the toolbox search path when you invoke the Pyxis Project Manager application by using this switch.

- **-version**

Version: Returns the version number of the `dmgr_ic` binary file. Typing `dmgr_ic` with the `-version` option does not invoke the Pyxis Project Manager application.

Return Values

VOID.

Description

The **dmgr_ic** command invokes the Pyxis Project Manager application in a shell window. You can then use the Pyxis Project Manager application interactively to manage your data design objects and tools. If you specify the `-nodisplay` option, the Pyxis Project Manager application invokes in the background.

Examples

1. This example invokes the Pyxis Project Manager application in a shell window.

dmgr_ic

2. This example invokes the Pyxis Project Manager application in nodisplay mode and then supplies a series of AMPLE functions.

```
dmgr_ic -nodisplay <<!  
> \SAMPLE_function_1();  
> \SAMPLE_function_2();  
> .  
..  
> !
```

The command syntax is not available when using nodisplay mode and backslashes are necessary to prevent the shell from interpreting the dollar signs in function names.

3. This example invokes the Pyxis Project Manager application in nodisplay mode and then supplies the pathname of an executable script.

```
dmgr_ic -nodisplay < $PROJ_XYZ/proj/d1/input_file
```


freeze_version

Freezes a version of a design object, preventing it from being deleted due to version depth.

Usage

```
freeze_version [options] {object_specifier}
```

Arguments

- **object_specifier**

Identifies a list of design objects and version numbers to be frozen. If the argument includes a type specifier, the list contains just the one specified design object. If the argument has no type specifier, the list contains all the types of design objects that have the given name. If the freeze_version command cannot freeze the specified versions of all listed design objects, it freezes as many as it can, but returns an error.

- **-h**

Help—displays online help and exits the command.

- **-s**

Silent—suppresses the display of all messages, including error messages. This option does not affect the error condition.

Return Values

VOID.

Description

The freeze_version command freezes the specified design object versions. Freezing a version prevents it from being deleted when it falls outside of the version depth of the design object.

Because the freeze_version command works on single versions of each specified design object, you can use version specifiers in the object_specifier argument. If you omit the version specifier, the freeze_version command freezes the current version.

If the specified design object is unversioned or if the specified version does not exist, the freeze_version command returns an error. If the specified version is already frozen, the freeze_version command does nothing, but does not return an error.

Examples

1. This example freezes version 2 of the “my_obj” design object of “my_type” type in the current working directory.

```
freeze_version my_obj:my_type[2]
```

2. This example freezes the current version of all design objects of “my_type” type in the directory at *\$PROJ_X/proj*.

```
list_contents -c -t my_type $PROJ_X/proj | freeze_version
```

Related Topics

[list_contents](#)

[revert_version](#)

[set_version_depth](#)

[unfreeze_version](#)

get_hard_name

Converts a list of pathnames to hard pathnames.

Usage

```
get_hard_name [options] {pathname_list}
```

Arguments

- **pathname_list**
The list of pathnames to convert. The command reads pathnames from standard input if you omit this list on the command line.
- **-h**
Help—displays online help and exits the command.
- **-s**
Silent—suppresses the display of all messages, including error messages. This option does not affect the error condition.

Return Values

VOID.

Description

This command converts a list of pathnames to equivalent hard pathnames, using the mechanisms of the DDMS pathname services. Pathnames beginning with a soft prefix are converted using the default location map (indicated by the value of the MGC_LOCATION_MAP environment variable), or by shell environment variables if no map is being used. Relative pathnames are converted using the MGC working directory (indicated by the value of the MGC_WD environment variable). The command lists the equivalent hard paths on standard output.

This command returns zero if it encounters no errors. Otherwise, it returns one and displays any error messages on standard output.

Examples

This example assumes you have a *\$DESIGN1* soft prefix that maps to the */project/x/design1* hard pathname. You need to escape the “\$” character at the beginning of a soft pathname, as shown by the “\” in the example.

```
get_hard_name \$DESIGN1/nand
```

Sample of the returned value:

```
/project/x/design1/nand
```

Related Topics

[get_soft_name](#)

get_soft_name

Converts a list of pathnames to soft pathnames.

Usage

`get_soft_name` [*options*] {*pathname_list*}

Arguments

- **pathname_list**
The list of pathnames to convert. The command reads pathnames from standard input if you omit this list on the command line.
- **-h**
Help—displays online help and exits the command.
- **-s**
Silent—suppresses the display of all messages, including error messages. This option does not affect the error condition.

Return Values

VOID.

Description

This command converts a list of pathnames to equivalent soft pathnames, using the mechanisms of the DDMS pathname services. Pathnames are converted using the soft prefixes in the default location map (indicated by the value of the `MGC_LOCATION_MAP` environment variable). Relative pathnames are converted using the MGC working directory (indicated by the value of the `MGC_WD` environment variable). The command lists the equivalent soft paths on standard output.

This command returns zero if it encounters no errors. Otherwise, it returns one and displays any error messages on standard output.

Examples

This example assumes you have a `$DESIGN1` soft prefix that maps to the `/project/x/design1` hard pathname.

```
get_soft_name /project/x/design1/nand
```

Sample of the returned value:

```
$DESIGN1/nand
```

Related Topics

[get_hard_name](#)

list_contents

Lists the design objects in a directory or a list of directories.

Usage

`list_contents [options] {directory_name}`

Arguments

- **directory_name**
Identifies the directory or directories whose contents are to be listed.
- **-a**
All versions—lists all versions of each design object and adds a version specifier to each object specifier in the output list. Unversioned objects are listed as version 1.
- **-c**
Current version—lists the current version of each design object and adds a version specifier to each object specifier in the output list. Unversioned objects are listed as version 1.
- **-f**
Full pathname—uses a full absolute pathname (starting with “/”) as the name of each object specifier in the output list.
- **-h**
Help—displays online help and exits the command.
- **-s**
Silent—suppresses the display of all messages, including error messages. This option does not affect the error condition.
- **-t type_name**
Type—lists only design objects of type `type_name`. To get an output listing that contains multiple types, you repeat this option.

Return Values

VOID.

Description

The `list_contents` command displays a list of all design objects within the specified directory or directories. Each item in the displayed list is an object specifier. When you issue this command with no options, the `list_contents` command displays the design objects in the specified directories and their types. By default, the `list_contents` command does not display the object's version number.

The `list_contents` command has two main uses:

1. You can use it to determine the name and type of the design objects in a directory or set of directories.
2. You can use its output (object specifiers) as the input to other design management shell commands. By directing the output of the `list_contents` command to a UNIX pipe. As a result, the output list of the `list_contents` command is available to the other command on its standard input stream. To customize the output list to be compatible with the other design management commands, you use the various options provided with the `list_contents` command.

Examples

Each example assumes that the current working directory is at `$PROJECT_XYZ/project` and that the *objects* sub-directory contains:

- The “my_obj” design object of “my_type” type, which includes versions 3, 4, and 5.
- The “his_obj” design object of “his_type” type, which is unversioned.

1. This example shows the default listing.

```
list_contents objects
```

The preceding example has the following output:

```
objects/my_obj:my_type
objects/his_obj:his_type
```

2. This example lists the current version of each design object.

```
list_contents -c objects
```

The preceding example has the following output:

```
objects/my_obj:my_type[5]
objects/his_obj:his_type[1]
```

3. This example lists all versions of each design object.

```
list_contents -a objects
```

The preceding example has the following output:

```
objects/my_obj:my_type[5]
objects/my_obj:my_type[4]
objects/my_obj:my_type[3]
objects/his_obj:his_type[1]
```

4. This example lists all versions and the full pathnames of each design object.

```
list_contents -a -f objects
```


The preceding example has the following output:

```
$PROJECT_XYZ/project/objects/my_obj:my_type[5]  
$PROJECT_XYZ/project/objects/my_obj:my_type[4]  
$PROJECT_XYZ/project/objects/my_obj:my_type[3]  
$PROJECT_XYZ/project/objects/his_obj:his_type[1]
```

Related Topics

[list_references](#)

listref

Lists the references of a design object.

Usage

`listref [options] {pathnames}`

Arguments

- **pathnames**
Identifies the pathname to the design object whose references are to be listed. Each directory can be specified as either an absolute or a relative pathname.
- **-h**
Help—displays online help.

Return Values

VOID.

Description

The **listref** command displays a list of references of the specified design objects. Each item in the displayed list is an object specifier.

If you specify the **pathnames** argument as a relative pathname, it is resolved using the value of the **MGC_WD** environment variable, if it valid; a message displays the relative pathname and the new resolved pathname. If the value of **MGC_WD** is not valid, the current working directory is used to resolve the relative pathname.

The **listref** command displays which version of the referenced object each reference targets. If the **listref** command cannot list the references of the specified versions of all listed design objects, it lists as many as it can, but returns an error.

Examples

This example lists the references held by the “my_obj” design object.

```
listref my_obj
```

The output includes the type and the version of the design object to which each reference points, as shown in the following sample:

```
$PROJECT_XYZ/design1/component:Mgc_container[5]  
$PROJECT_XYZ/doc/funct:Mgc_file[1]  
$PROJECT_XYZ/design2/schematic3:Mgc_container[2]
```

Related Topics

[chref](#)

[change_references](#)

list_references

Lists the references of a design object and displays the name and type of the referenced object(s).

Usage

`list_references` [*options*] {*object_specifier*}

Arguments

- **object_specifier**

Identifies a list of design objects and version numbers whose references are to be listed. If the argument includes a type specifier, the list contains just the one specified design object. If the argument has no type specifier, the list contains all the types of design objects that have the given name. If the `list_references` command cannot list the references of the specified versions of all listed design objects, it lists as many as it can, but returns an error.

- **-h**

Help—displays online help and exits the command.

- **-l**

Long—displays a detailed listing that includes the following items:

- reference handle, the index of the reference within the design object
- version of the referenced design object
- reference state: fixed, current, or read-only
- type of the pathname: hard, soft, or non-file system
- type of the referenced design object
- pathname of the referenced design object

- **-m**

Mapped pathnames—reference pathnames are converted from soft pathnames to hard pathnames. If the reference pathname cannot be converted, it is stored in the reference as shown.

- **-p**

Properties—lists the properties held by each reference. To use this option, you must also use the **-l** (long) option; otherwise, this option does nothing.

- **-s**

Silent—suppresses the display of all messages, including error messages. This option does not affect the error condition.

- **-v**

Version—adds a version specifier to each reference pathname (object specifier) in the output list. To use this option, you must omit the **-l** (long) option; otherwise, this option does nothing.

Return Values

VOID.

Description

Each item in the displayed list is an object specifier. When you issue this command with no options, the `list_references` command displays the name and type of each referenced design object, including its absolute pathname. By default, the `list_references` command does not display the version of the referenced object that each reference cites.

The `list_references` command has two main uses:

1. You can use it to determine the references (and information about those references) that a design object or set of design objects hold.
2. You can use its output (object specifiers) as the input to other design management shell commands, by directing the output of the `list_references` command to a UNIX pipe. As a result, the output list of the `list_references` command is available to the other command on its standard input stream. To customize the output list to be compatible with the other design management commands, you use the various options provided with the `list_references` command. Some of the options apply only when viewing the output list directly.

Because the `list_references` command can list the references of each version of the specified design object, you can use version specifiers in the *object_specifier* argument.

Examples

1. This example lists the references held by version 3 of the “my_obj” design object of “my_type” type.

```
list_references -v my_obj:my_type[3]
```

The output includes the version of the design object to which each references points, as shown below:

```
her_obj:her_type[1]
his_obj:his_type[1]
my_obj:my_type[2]
```

2. This example deletes all design objects referenced by the current version of the “my_obj” design object of “my_type” type.

```
list_references my_obj:my_type | delete_object
```

Related Topics

[change_references](#)

[list_contents](#)

move_object

Moves a design object to a new location.

Usage

`move_object [options] {src_object_specifier dest_object_specifier}`

Arguments

- **src_object_specifier**

Identifies a list of design objects to be moved. If the argument includes a type specifier, the source list contains just the one specified design object. If the argument has no type specifier, the source list contains all the types of design objects that have the given name. If the `move_object` command cannot move all the design objects on the source list, it moves as many as it can, but returns an error.

- **dest_object_specifier**

Identifies a list of design objects to which the source list is to be moved. If the argument includes a type specifier, then the destination list contains just the one specified design object. If the argument has no type specifier, the destination list contains all the types of the design objects that are on the source list. For each type of design object on the source list, the destination list must contain an identical type of design object; otherwise, the `move_object` command returns an error.

If a source design object already exists at the destination, the `move_object` command returns an error, unless the replacement switch **-r** is specified.

- **-h**

Help—displays online help and exits the command.

- **-r**

Replace—replaces an existing destination design object with the specified source design object. If no design object already exists at the destination, the `move_object` command ignores this option.

- **-s**

Silent—suppresses the display of all messages, including error messages. It does not affect the error condition.

Return Values

VOID.

Description

The `move_object` command moves a design object, including all of its versions, from the specified source location to the specified destination. If the source design object is a directory or a container (that is, it has a directory as a fileset member), the `move_object` command also

moves the contents of the directory. If some version or versions of the source design object are frozen, the `move_object` command leaves the same destination design object versions frozen.

If a design object already exists at the destination location, you must use the **-r** (replacement) option; otherwise, the `move_object` command returns an error.

The references of the destination design object are the same as those of the source design object. That is, they might still point to the source design object or its contents, although they no longer exist. In order to repair these or other references to reflect the new location, you use the `change_references` command.

Because the `move_object` command moves each specified design object as a whole, you must not use version specifiers in the `src_object_specifier` and `dest_object_specifier` arguments. If you do, the `move_object` command returns an error.

Caution

If you omit the type specifier portion of *object_specifier*, `move_object` attempts to move every design object of any type whose name matches the name specifier that you provide. This may result in some non-design object files or directories (file system objects that are not part of a recognized design object) with the same name being moved, along with the design objects you intended to move. To avoid inadvertently moving files and directories, you should always provide the type specifier along with the name specifier to the `move_object` shell command.

If you omit the type specifier portion of *src_object_specifier*, `move_object` attempts to move every design object whose name matches the name specifier that you provide. If the type of one of these source design objects is not loaded into the current process' type manager, `move_object` moves the fileset members of that object as ordinary files and directories. This creates a corrupt design object at the destination, and leaves a corrupt design object at the source.

To prevent this from happening, either always use a type specifier, which causes `move_object` to display an error message for an unrecognized type, or carefully verify that the source object's type is loaded. For instructions on how to verify that an object's type is loaded, refer to [“Finding the Name and Type of a Design Object in the Shell”](#) on page 724.

Examples

This example moves the design object “my_obj” of “my_type” type in the current working directory to the “his_obj” design object of my_type, also in the current working directory.

```
move_object -r my_obj:my_type his_obj
```

If the current working directory already contains the his_obj design object of “his_type” type, this command replaces that object with my_obj.

Related Topics

[change_references](#)

[list_contents](#)

[copy_object](#)

[freeze_version](#)

[unfreeze_version](#)

revert_version

Deletes the current version of a design object and makes the previous version current.

Usage

`revert_version [options] {object_specifier}`

Arguments

- **object_specifier**

Identifies a list of the design objects whose current version is to be deleted. If the argument includes a type specifier, the list contains just the one specified design object. If the argument has no type specifier, the list contains all the types of design objects that have the given name. If the `revert_version` command cannot delete the current version of all listed design objects, it deletes as many as it can, but returns an error.

- **-h**

Help—displays online help and exits the command.

- **-s**

Silent—suppresses the display of all messages, including error messages. This option does not affect the error condition.

Return Values

VOID.

Description

The `revert_version` command deletes the current version of the specified design object or objects, and makes their previous version current. If the design object is unversioned, it has only one version, or its current version is frozen, then the `revert_version` command returns an error.

Because the `revert_version` command works on the current version of each specified design object, it treats each design object as a whole object. If you use version specifiers in the *object_specifier* argument, the `revert_version` command returns an error.

Examples

1. This example deletes the current version of the design object “my_obj” of “my_type” type in the current working directory and makes the previous version current.

```
revert_version my_obj:my_type
```

2. This example performs revert version on all design objects in the directory at *\$PROJECT_XYZ/project*.

```
list_contents $PROJECT_XYZ/project | revert_version
```

Related Topics

[freeze_version](#)

[list_contents](#)

[set_version_depth](#)

[unfreeze_version](#)

salvage_object

Salvages a design object damaged by an application failure.

Usage

salvage_object [*options*] {object_specifier}

Arguments

- **object_specifier**

Identifies a list of design objects to be salvaged. If the argument includes a type specifier, then the list contains just the one specified design object. If the argument has no type specifier, the list contains all the types of design objects that have the given name. If the salvage_object command cannot salvage all design objects on the list, it salvages as many as it can, but returns an error.

- **-a**

Attribute file—verifies that the format and content of the specified design object's attribute file is correct.

- **-h**

Help—displays online help and exits the command.

- **-r**

Recover—tries to recover data from temporary edit files. These files might exist if an application fails while you are editing the design object. If possible, the salvage_object command creates a new version of the design object from the temporary edit files. In any case, the salvage_object command deletes the temporary edit files. If you omit this option, the salvage_object command deletes all of these temporary edit files without salvaging their contents.

- **-s**

Silent—suppresses the display of all messages, including error messages. This option does not affect the error condition.

Return Values

VOID.

Description

The salvage_object command repairs a design object that has been damaged through an application failure. Normally, when you close a design object after editing it, the application removes the temporary lock (*.lck*) and edit (*.ed*) files used during the edit session. However, if the application fails while you are editing the design object, these temporary files might still exist on disk. The salvage_object command removes all these temporary files.

Because the `salvage_object` command treats each specified design object as a whole, you cannot use version specifiers in the *object_specifier* argument. If you do, the `salvage_object` command returns an error.

Examples

This example salvages the design object “my_obj” of “my_type” type in the current working directory.

```
salvage_object my_obj:my_type
```

Related Topics

[list_contents](#)

set_version_depth

Specifies the number of versions that the specified design object or objects keep.

Usage

```
set_version_depth [options] {object_specifier}
```

Arguments

- **object_specifier**

Identifies a list of the design objects whose version depth is to be set. If the argument includes a type specifier, the list contains just the one specified design object. If the argument has no type specifier, the list contains all the types of design objects that have the given name. If the `set_version_depth` command cannot set the version depth of all listed design objects, it sets as many as it can, but returns an error.

- **-d *version_depth***

Depth—indicates the version depth to be set. If you omit the **-d** option, the `set_version_depth` command uses the default version depth for the design object type specified by *object_specifier*.

- **-h**

Help—displays online help and exits the command.

- **-s**

Silent—suppresses the display of all messages, including error messages. This option does not affect the error condition.

Return Values

VOID.

Description

If the design object is unversioned, the `set_version_depth` command returns an error.

If you set the version depth lower than the previous version depth of the design object, the `set_version_depth` command deletes all unfrozen versions that are outside the version depth. It never deletes frozen versions.

Because the `set_version_depth` command treats each specified design object as a whole, you cannot use version specifiers in the *object_specifier* argument. If you do, the `set_version_depth` command returns an error.

Examples

1. This example sets the version depth of the design object “my_obj” of “my_type” type in the current working directory to 3.

```
set_version_depth -d 3 my_obj:my_type
```

2. This example sets the version depth of all design objects in the directory at *\$PROJECT_XYZ/proj* to three.

```
list_contents $PROJECT_XYZ/proj | set_version_depth -d 3
```

Related Topics

[freeze_version](#)

[list_contents](#)

[revert_version](#)

[unfreeze_version](#)

show_object_info

Displays information for the specified design object or objects.

Usage

```
show_object_info [options] {object_specifier}
```

Description

The `show_object_info` command displays information for the specified design object or objects. This information includes version history, lock information, fileset members, references, and properties. This command also displays object properties, properties held only by a specific version, and properties held by references.

Arguments

- **object_specifier**
Identifies a list of design objects and version numbers whose information is to be listed. If the argument includes a type specifier, the list contains just the one specified design object. If the argument has no type specifier, the list displays information about all types of design objects that have the given name. If the `show_object_info` command cannot provide information for the specified versions of all listed design objects, it displays information about as many as it can, but returns an error.
- **-h**
Help—displays online help and exits the command.
- **-m**
Mapped pathnames—reference pathnames are converted from soft to hard pathnames. If the pathname cannot be converted, it is stored in the reference as shown.
- **-s**
Silent—suppresses the display of all messages, including error messages. This option does not affect the error condition.

Return Values

VOID.

Examples

1. This example displays information for the current version of the design object “my_obj” of “my_type” type.

```
show_object_info my_obj:my_type
```

2. This example shows the information listed for “my_config”.

```
show_object_info my_config
```

```
-----
Information for object my_config
-----
Full pathname: $PROJ_XYZ/proj/my_config
Type: Dme_config_do
Current version: 2
Version depth: -1
Object complete: YES
Fileset size: 1
Date last modified: Wed May  1 11:09:28 1991
Object is locked
User: susanp
On host: susanp
Time of lock: Wed May  1 11:05:18 1991
Current version of object has 2 object properties.
Name                      Value
----                      -
writer                    bob
engineer                  mary
Versions present (2):
Fileset members for version 1
my_config.cfg_1           File
Fileset members for version 2
my_config.cfg_2           File
References for $PROJ_XYZ/proj/my_config Dme_config_do (Ver 2)
handle  version state    object type    object pathname
-----  -
0       1       current  Mgc_container  $PROJ_XYZ/proj/fig
1       1       current  Mgc_file       $PROJ_XYZ/proj/zoo
2       1       current  Dme_config_do  $PROJ_XYZ/proj/pie
```

Related Topics

[list_contents](#)

[list_references](#)

unfreeze_version

Unfreezes a frozen version of a design object.

Usage

`unfreeze_version` [*options*] {*object_specifier*}

Arguments

- **object_specifier**

Identifies a list of design objects and version numbers to be unfrozen. If the argument includes a type specifier, the list contains just the one specified design object. If the argument has no type specifier, the list contains all the types of design objects that have the given name. If the `unfreeze_version` command cannot unfreeze the specified versions of all listed design objects, it unfreezes as many as it can, but returns an error.

- **-h**

Help—displays online help and exits the command.

- **-s**

Silent—suppresses the display of all messages, including error messages. It does not affect the error condition.

Return Values

VOID.

Description

The `unfreeze_version` command unfreezes the specified design object versions. If the newly unfrozen version falls below the version depth of the design object, that version is not deleted. However, if left in the unfrozen state, the version is deleted when you set the version depth again or create another version of the design object.

Because the `unfreeze_version` command works on single versions of each specified design object, you can use version specifiers in the *object_specifier* argument. If you omit the version specifier, the `unfreeze_version` command unfreezes the current version.

If the specified design object is unversioned or the specified version does not exist, the `unfreeze_version` command returns an error. If the specified version is not already frozen, the `unfreeze_version` command does nothing, and it does not return an error.

Examples

This example unfreezes version 2 of the design object “my_obj” of “my_type” type in the current working directory.

```
unfreeze_version my_obj:my_type[2]
```

Related Topics

[freeze_version](#)

[list_contents](#)

[revert_version](#)

[set_version_depth](#)

Appendix A

AMPLE Data Types

The Pyxis Project Manager application uses common data types.

Common Data Types 779

Common Data Types

This table lists all the data types available in the Pyxis Project Manager application.

Table A-1. Common Data Types

Argument Constraint	Description
Boolean	A name literal that has one of two values: @true or @false.
Callable	A function name.
Complex	A complex number in the format [real_part, imaginary_part].
Enum	A vector of zero or more numbers in the format [number1, ..., numberN].
Form	The location in memory of a dialog box, including the dialog box site and gadgets.
Form_gadget	The location in memory of a dialog box gadget.
Form_site	The location in memory of a dialog box site, including gadgets.
Form_site_value	The location in memory of a dialog box site.
Integer	A number that must be an integral number.
Location	A location in the format [number, number, string], where the last element is optional.
Menu	The location in memory of a menu, including menu items.
Menu_item	The location in memory of a menu item.
Name	A name literal value. In function usage each name value is preceded by an at symbol “@”. In command usage, omit the at symbol.
Number	A number, including integer, real, or scientific notation.


Table A-1. Common Data Types (cont.)

Argument Constraint	Description
Pathname	A string that specifies the location of a file in the directory hierarchy. The pathname must be in the format required by your workstation.
Polylocation	A vector of zero or more locations in the format [location1, ..., locationN].
Polyrectangle	A vector of zero or more rectangles in the format [rectangle1, ..., rectangleN].
Real	A number that is a floating point number.
Rectangle	A two element vector of locations in the format [location1, location2].
Status	A data object that consists of a numeric value and additional parameters.
String	A value that must be quoted with single or double quotes if it begins with an up-arrow (^) or a square bracket ([), or it contains commas, white space, semicolons, or pound signs (#).
Vector	A collection of zero or more values or vectors in the format [element1, ..., elementN].

Appendix B

Toolkit Examples

The following topics provide examples from each of the Pyxis Project Manager toolkits illustrating the respective function usage.

Note  You should not attempt to execute these examples; they do not generate usable design objects.

For more information about the functions used in the examples, refer to “[Function Dictionary Part 1](#)” on page 43.

Design Object Toolkit Examples	781
Configuration Management Toolkit Examples.....	783
Qualification Script Example	787

Design Object Toolkit Examples

The following example demonstrates the use of the design object toolkit functions that lock, save, and unlock a design object.

The example performs the following operations:

1. Locks a design object.
2. Annotates the design object with: object properties, version properties, references, and reference properties.
3. After each operation, calls a second function that checks for errors and prints the error code. If no error code is found, prints an “OK” message in the transcript window.
4. Saves and unlocks the object.

The example assumes that the following objects of “Document_file” type exist:

- *\$MGC_HOME/tmp/cities/new_york*
- *\$MGC_HOME/tmp/cities/los_angeles*

```
function $test_function()
{
    local la = "$MGC_HOME/tmp/cities/los_angeles";
    local ny = "$MGC_HOME/tmp/cities/new_york";
    local d_type = "Document_file";
    // Lock the object new_york in annotate mode.
    $$lock_object(ny, d_type, @annotate);
    $dump_status("$$lock_object");
    // Add a reference to the current version.
    $$add_reference( ny, d_type, , la, d_type);
    $dump_status("$$add_reference");
    // Add an object property to the current version.
    $$set_object_property(ny,d_type, , "state", "New York");
    $dump_status("$$set_object_property");
    // Add a version property to the current version.
    $$set_version_property(ny,d_type, , "population", "17
        million");
    $dump_status("$$set_version_property");

    // Add a reference property to the current version.
    $$set_reference_property(ny,d_type, ,la,d_type, ,
        "distance", "3000 miles");
    $dump_status("$$set_reference_property");
    // Save and unlock the design object.
    $$save_object(ny, d_type);
    $dump_status("$$save_object");
    $$unlock_object(ny, d_type);
    $dump_status("$$unlock_object");
}
```

The following example of the \$dump_status() function prints out the status codes. If the status code stack is empty, the function prints an “OK” message to the transcript window.

```
function $dump_status(s:string)
{
    local v;
    local message;
    local st = $$get_status_code();
    if (st != 0) {
        message = $strcat("Bad status returned by ", s,
            " <", st, ">\n");
        $writes_file($stdout(), message);
        v = $$get_status_messages();
        local count = 0;
        local len = length(v);
        while (count < len) {
            $writes_file($stdout(), $strcat("<<message "
                , count, ">>" , v[count], "\n" ));
            count = count + 1;
        }
    }
    else {
        message = $strcat(s, " returned ok status\n");
        $writes_file($stdout(), message);
    }
    return st;
}
```

Related Topics

[Design Object Toolkit Functions](#)

[AMPLE for Pyxis Reference Manual](#)

Configuration Management Toolkit Examples

The following example builds a configuration by using functions from the configuration management toolkit.

The example performs the following operations:

1. Builds the configuration relative to the current home directory
2. Adds the current version of two design objects, and a previous version of another
3. Sets the build rules for one of the primary entries
4. Builds the configuration
5. Removes a secondary entry and sets the target path of another secondary entry
6. Copies the configuration to a target location
7. Saves and closes the configuration

This example assumes that the following design objects exist in the “*example_dir*” directory within the user's home directory:

- “versioned_obj” of “Pcb_notes” type
- *dir_a* and *dir_b* of “Mgc_container” type
- “dir_b” contains a file named *file_e*


```

function create_and_copy_an_object()
{
// Create a configuration in the current home directory.
local test_config_path = $strcat($get_env("HOME"),
    "/example_dir/example_tk_config");
$$create_configuration(test_config_path);
// Check status code, if not 0 print result.
if ($$get_status_code() != 0) {
    $$report_global_status();
}
// Add configuration entries using relative pathnames.
local dir_a_path = $strcat($get_env("HOME"),
    "/example_dir/dir_a");
local dir_b_path = $strcat($get_env("HOME"),
    "/example_dir/dir_b");
local versioned_obj_path = $strcat($get_env("HOME"),
    "/example_dir/versioned_obj");
// Add the current version of dir_a and dir_b.
$$add_configuration_entry(dir_a_path, "Mgc_container", 0);
$$add_configuration_entry(dir_b_path, "Mgc_container", 0);

// Add version 2 of a versioned object.
$$add_configuration_entry(versioned_obj_path, "Pcb_notes", 2);

// Set reference traversal to "off" for dir_a.
$$set_reference_traversal(@off, dir_a_path,
    "Mgc_container", 0);

// Set the filters on dir_a_path.
local path_filter_criteria = [["dir_a"],["exclude"]];
$$set_object_path_filter(path_filter_criteria, dir_b_path,
    "Mgc_container", 0);
local type_filter_criteria = [["Mgc_file"],["include"]];
$$set_object_type_filter(type_filter_criteria, dir_b_path,
    "Mgc_container", 0);
local object_prop_criteria =
    [["name_2"],["false"],["exclude"]];
$$set_object_property_filter(object_prop_criteria,
    dir_b_path, "Mgc_container", 0);
local reference_prop_criteria =
    [["name_1"],["false"],["exclude"]];
$$set_reference_property_filter(reference_prop_criteria,
    dir_b_path, "Mgc_container", 0);
// Build the configuration.
$$build_configuration();
// check status code, if not ok print result.
if ($$get_status_code() != 0) {
    $$report_global_status();
}

// Modify/Remove configuration entries.
// Reset the target of file_e (a secondary of dir_b).
local file_d_path =
    $strcat($get_env("HOME"), "/example_dir/file_d");
$$set_target_path("dir_a/file_z", file_d_path,
    "Mgc_file", 0);
// Remove a design object.
local file_b3_path = $strcat($get_env("HOME"),

```

```
"/example_dir/dir_b/file_b3");
$$remove_configuration_entry(file_b3_path, "Mgc_file", 0);

// Copy the configuration.
local target_path = $strcat($get_env("HOME"),
    "/example_dir/copy_output_dir");
$$copy_configuration(target_path, @cancel, @create);
// Print the status of the configuration.
$print_overall_config_status();

// Save and Close the Configuration.
$$save_configuration();
$$close_configuration();
}
```

The following example of the `$print_overall_config_status()` function prints out the status of the configuration object. It uses several of the configuration management query functions, and the AMPLE `$writeln_file()` and `$strcat()` functions, and `$stdout`.

```
function $print_overall_config_status()
{
    // Print the configuration's path.
    local configuration_path = $$get_configuration_path();
    if (configuration_path != undefined) {
        $writeln_file($stdout,$strcat(" Configuration Path = ",
            configuration_path));
    }
    else {
        $writeln_file($stdout," $$get_configuration_path
            returned undefined");
        return;
    }
    // Check if build is valid, print result.
    local build_valid = $$is_build_valid();
    $writeln_file($stdout,$strcat(" Build Valid = ",
        build_valid));
    // Check if build is consistent, print result.
    local build_consistent = $$is_build_consistent();
    $writeln_file($stdout,$strcat(" Build Consistent = ",
        build_consistent));
    // Print out each primary entry using a loop.
    local primaries = $$get_primaries();
    $writeln_file($stdout," List Primary Entries");
    local i;
    if ($type(primaries,"vector") && (length(primaries) > 0))
    {
        for ( i = 0; i < length(primaries) ; i = i + 1 ) {
            $writeln_file($stdout, primaries[i] );
        }
    }
    // Print all of the entries using a loop.
    local config_entries = $$get_configuration_entries();
    $writeln_file($stdout," List Configuration Entries");
    if ($type(config_entries,"vector") &&
        (length(config_entries) > 0)) {
        for ( i = 0; i < length(config_entries) ; i = i + 1 ) {
            $writeln_file($stdout, config_entries[i] );
        }
    }
}
```

Related Topics

[Configuration Management Toolkit Examples](#) *AMPLE for Pyxis Reference Manual*

Qualification Script Example

The following example is a Decision Support System (DSS) qualification script that uses several functions available from the Pyxis Common User Interface.

The example creates a dialog box containing predefined values.

```
//  
// DSS.QUAL -- for Pyxis Project Manager  
//  
  
function $dss(sheet      : optional string {default = ""},  
    startflag    :      string {default = ""},  
    startupfile  : optional string {default = ""},  
    installflag  :      string {default = ""},  
    installfile  : optional string {default = ""},  
    titleflag    :      string {default = ""},  
    titlestr     : optional string {default = ""},  
    pkgflag      :      string {default = ""},  
    pkgstr       : optional string {default = ""},  
    geomflag     :      string {default = ""},  
    geomstr      : optional string {default = ""},  
    displayflag  :      string {default = ""},  
    displaystr   : optional string {default = ""},  
    switcharg    : optional), INDIRECT  
  
{  
    $message($format("dss %s %s %s %s %s %s %s %s %s %s %s %s %s  
        %s %s", sheet, startflag, startupfile, installflag,  
        installfile, titleflag, titlestr, pkgflag,  
        pkgstr, geomflag, geomstr, displayflag,  
        displaystr), @note);  
    $invoke_tool("$MGC_HOME/bin/dss", sheet, startflag,  
        startupfile, installflag, installfile, titleflag,  
        titlestr, pkgflag, pkgstr, geomflag, geomstr,  
        displayflag, displaystr);  
}  
  
function $ok_cancel_site(), INVISIBLE  
{  
    return $form_row(@false,  
        $form_button(" OK ", "$execute()", @true),  
        $form_button(" Cancel ", "$forget()"));  
}  
  
function $dss_form(), INVISIBLE  
{  
    local c1 = $form_right_justified_column(@false,  
        // Geometry Switch  
        $form_switch(@buttons, 9, "Specify Geometry:",  
            [$form_item("No", "")],  
            $form_row(@false, $form_argument_value(9, ""),  
                $form_argument_value(10, ""))),  
            [$form_item("Yes", "-geometry"),  
                $form_row(@false, $form_argument_value(9, "-  
geometry"),  
                    $form_argument_gadget(10, $form_string_entry_box_gadget  
("Geometry:")))]),  
        // Display Switch  
        $form_switch(@buttons, 11, "Specify Display:",  
            [$form_item("No", "")],  
            $form_row(@false, $form_argument_value(11, ""),  
                $form_argument_value(12, ""))),  
            [$form_item("Yes", "-display"),  
                $form_row(@false, $form_argument_value(11, "-
```

```

        display"),
        $form_argument_gadget(12,$form_string_entry_box_gadget(
"Display:")))]));
local c2 = $form_right_justified_column(@false,
// Custom Title Switch
$form_switch(@buttons,5,"Specify Custom Title:",
[$form_item("No", ""),
$form_row(@false,$form_argument_value(5,""),
$form_argument_value(6,""))],
[$form_item("Yes", "-title"),
$form_row(@false,$form_argument_value(5,"-
title"),
$form_argument_gadget(6,$form_string_entry_box_gadget
("Title:")))]),
// Custom Package Switch
$form_switch(@buttons,7,"Custom Package:",
[$form_item("No", ""),
$form_column(@false,$form_argument_value(7,""),
$form_argument_value(8,""))],
[$form_item("Yes", "-package"),
$form_column(@false,$form_argument_value(7,"-
package"),
$form_argument_gadget(8,$form_string_entry_box_gadget("Packag
e:")))]));
local r1 = $form_row(@false, $form_label("DSS","helvetica-
bold:14"));
local r2 = $form_row(@true, $form_label($strcat("DSS
Object: ", "$get_object_pathname()")));
local r3 = $form_switch(@buttons,1,"Startup Option: ",
// Startup Switch
[$form_item("Default", ""),
$form_row(@false,$form_argument_value(1,""),
$form_argument_value(2,""))],
[$form_item("None", "-nostartup"),
$form_row(@false,$form_argument_value(1,"-
nostartup"),
$form_argument_value(2,""))],
[$form_item("Specify", "-startup"),
$form_row(@false,$form_argument_value(1,"-
startup"),
$form_argument(2,$form_gadget_value(
$form_string_entry_box_gadget("Pathname:")),400))]);
local r4 = $form_row(@true, c1, c2);
local r5 = $form_switch(@buttons,3,"Install Application:",
// Install Application Switch
[$form_item("No", ""),
$form_row(@false,$form_argument_value(3,""),
$form_argument_value(4,""))],
[$form_item("Yes", "-install"),
$form_row(@false,$form_argument_value(3,"-
install"),
$form_argument(4,$form_gadget_value(
$form_string_entry_box_gadget("Pathname:")),400))]);
local layout = $form_column(@true, r1, r2, r3, r4, r5,
$ok_cancel_site());

```

```
extern dss_form =
$create_form($qual_script_scope,@$dss,@true,"Wheat",layout);

$set_form_position(dss_form, 50, 100, 0, 450);
}

{
    switch ($get_object_type()) {
        case "mgc_dss_tool"      : { $dss(); break;}
        case "mgc_dss_object"   : { $dss($get_object_pathname());
break;}
        case "mgc_dsb_object"   : { $dss($get_object_pathname());
break;}
        case VOID               : { $dss(); break;}
        default                 : $message("Problem with
qualification script.",@note);
    }
}
```

Appendix C

Regular Expressions

The following topics provide a brief overview of the UNIX System V regular expressions and wildcards.

UNIX System V Regular Expressions	791
Wildcards	793

UNIX System V Regular Expressions

Regular expressions are a shorthand notation for expressing whole categories of strings that share a common rule of construction. The key idea is that the regular expression represents and matches all of the strings in the category. As a result, one regular expression can match many different strings.

Regular expressions are themselves strings of characters: some ordinary, some special. [Table C-1](#) presents the special characters by which you build regular expressions and the meaning of each.

Table C-1. UNIX System V Regular Expressions

Notation	Description
()	Groups multiple concatenated regular expressions into a single regular expression
.	A wildcard that matches any single character
[]	Matches any single character enclosed in the brackets
[^]	Matches any single character, except one enclosed by the brackets
-	Defines a range of characters when used within brackets
^	Matches the beginning of a string when used outside of brackets
\$	Matches the end of a string
*	Matches zero or more occurrences of the preceding regular expression
+	Matches one or more occurrences of the preceding regular expression
{m,n}	Matches X occurrences of the preceding regular expression, where 0 <= m <= X <= n <= 255
{m}	Matches m occurrences of the preceding regular expression
{m,}	Matches at least m occurrences of the preceding regular expression

Table C-1. UNIX System V Regular Expressions

Notation	Description
\	Turns the following special character into an ordinary character, which implies that it matches only itself and has no special meaning for matching

The functions listed in [Table C-2](#) accept UNIX System V regular expressions as arguments. In addition, the shell commands [chref](#) and [change_references](#) also accept regular expressions.

Table C-2. Pyxis Project Manager Functions Accepting Regular Expressions

Function Name
\$\$change_configuration_references()
\$change_configuration_references()
\$\$change_design_object_references()
\$change_design_object_references()
\$\$change_object_references()
\$change_object_references()
\$\$copy_object()
\$copy_object()
\$descend_hierarchy_specify_level()
\$\$duplicate_object()
\$\$get_object_path_filter()
\$\$get_object_type_filter()
\$\$release_object()
\$release_object()
\$search()
\$set_build_rules()
\$\$set_object_path_filter()
\$\$set_object_type_filter()
\$show_component_hierarchy()

Wildcards

Some functions in the Pyxis Project Manager application accept UNIX System V wildcards as arguments.

The wildcard characters listed in [Table C-3](#) apply to selecting and unselecting design objects.

Table C-3. Wildcards

Notation	Description
*	Matches zero or more occurrences of the preceding character, including the null string
?	Matches any single character zero or more times
[...]	Matches any one of the enclosed characters. A pair of characters separated by “-” matches any character between the inclusive pair. For example, [A-Z] matches all letters between A and Z. If the first character following the “[” is “!”, any character not enclosed in the brackets is matched.
\	Turns the following special character into an ordinary character, which implies that it matches only itself and has not special meaning for matching.

Note



You must match a leading “.” explicitly. The Pyxis Project Manager wildcards do not support full UNIX pathnames; therefore, there is no special handling of “/”.

The functions listed in [Table C-4](#) accept UNIX System V wildcards as arguments.

Table C-4. Pyxis Project Manager Functions Accepting Wildcards

Function Name
\$browse_for_object()
\$select_by_name()
\$select_by_type()
\$select_config_entry()
\$select_object()
\$select_reference()
\$select_tool()
\$select_toolbox()
\$select_trash_object()
\$select_version()

Table C-4. Pyxis Project Manager Functions Accepting Wildcards

Function Name
\$setup_filter_active()
\$setup_filter_all()
\$unselect_by_name()
\$unselect_by_type()
\$unselect_config_entry()
\$unselect_object()
\$unselect_reference()
\$unselect_tool()
\$unselect_toolbox()
\$unselect_trash_object()
\$unselect_version()

Appendix D

Function Cross-Reference

The following topics describe how Pyxis Project Manager commands and menu paths are mapped to their corresponding Pyxis Project Manager functions. This appendix also presents tables that map logical key names to their corresponding Pyxis Project Manager functions and scopes and to their corresponding physical keys on each of the Mentor Graphics-supported keyboards.

Commands to Functions	795
Menu Paths to Functions.....	796
Logical Key Names Mapped to Functions and Scopes.....	803
Logical Key Names Mapped to Physical Keys	806

Commands to Functions

Each command maps to exactly one function.

The name of the corresponding function is the same as that of the command, except that a dollar sign “\$” or a double dollar sign “\$\$” begins the function name, all spaces become underscores “_”, and the function name ends with a pair of open and close parentheses “()”.

For example, the Add Configuration Entry command maps to the \$add_configuration_entry() function.

Related Topics

[Logical Key Names Mapped to Functions and Scopes](#)

[Logical Key Names Mapped to Physical Keys](#)

[Menu Paths to Functions](#)

Menu Paths to Functions

The Pyxis Project Manager application also maps menu paths to its corresponding interactive functions.

The available menu items vary based on the context or active window. For instance, the Edit pulldown menu displays when the context is the Configuration window but is not available for the Component window.

In addition, within the same active window as context, some menu items are available only in the pulldown menu or popup menu, while others may be available in both. By default, the menu items are those available in the pulldown menu unless otherwise specified.

See [Table D-1](#) for details on the pulldown menu items and [Table D-2](#) for the popup menu item mapping to the respective Pyxis Project Manager commands.

Table D-1. Pulldown Menu Paths Mapped to Functions

Window	Pulldown Menu Path	Function
Any window	File > New	\$browse_for_object()
	File > Open	
	Setup > Admin Login	\$login_admin()
	Setup > Check Registries	\$check_registries()
	Windows > Open Configuration > New Existing	\$open_configuration_window()
	Windows > Open Navigator > View by Icon View by Name	\$open_navigator()
	Windows > Open Session Monitor	\$open_session_monitor()
	Windows > Open Tools Window	\$open_tools_window()
	Windows > Open Trash Window	\$open_trash_window()
	Windows > Open Types Window	\$open_types_window()
Configuration	Edit > Add Entry	\$add_configuration_entry()
	Edit > Add Versions	\$add_versions()
	Edit > Change References	\$change_configuration_references()
	Edit > Convert References	\$convert_configuration_references()
	Edit > Copy Configuration	\$copy_configuration()
	Edit > Delete Configuration	\$delete_configuration()
	Edit > Maintain Hierarchy	\$maintain_hierarchy()
	Edit > Release Configuration	\$release_configuration()

Table D-1. Pulldown Menu Paths Mapped to Functions (cont.)

Window	Pulldown Menu Path	Function
	Edit > Remove Entry	\$remove_configuration_entry()
	Edit > Set Build Rules	\$set_build_rules()
	Edit > Set Target Path	\$set_target_path()
	File > Build	\$build_configuration()
	File > Export Configuration	\$export_configuration_entries()
	File > Freeze	\$freeze_configuration()
	File > Lock Configuration	\$lock_configuration()
	File > Save	\$save_configuration()
	File > Save As	\$save_configuration_as()
	File > Select > All	\$select_all()
	File > Select > By Name	\$select_by_name()
	File > Select > By Name	\$select_by_name()
	File > Select > By Name	\$select_by_name()
	File > Select > By Type	\$select_by_type()
	File > Unfreeze	\$unfreeze_configuration()
	File > Unlock Configuration	\$unlock_configuration()
	File > Unselect > All	\$unselect_all()
	File > Unselect > By Name	\$unselect_by_name()
	File > Unselect > By Type	\$unselect_by_type()
	Report > Report Configuration Info	\$report_configuration_info()
	Report > Entry Info	\$report_entry_info()
	Report > Entry Verification	\$report_entry_verification()
	Report > References	\$report_configuration_references()
	View > Containment Hierarchy	\$view_containment_hierarchy()
	View > Hide Secondary Entries	\$hide_secondary_entries()
	View > Primary Hierarchy	\$view_primary_hierarchy()
	View > Secondary Entries	\$view_secondary_entries()
Navigator	Edit > Change > Name	\$change_object_name()
	Edit > Change > Property	\$change_object_property()
	Edit > Change > Protection	\$change_protection()

Table D-1. Pulldown Menu Paths Mapped to Functions (cont.)

Window	Pulldown Menu Path	Function
	Edit > Change > References	\$change_design_object_references() \$change_object_references()
	Edit > Change > Version Depth	\$change_version_depth()
	Edit > Check References	\$check_references()
	Edit > Convert References	\$convert_object_references()
	Edit > Copy To	\$copy_design_object() \$copy_object()
	Edit > Delete Object > Excess Versions	\$delete_excess_versions()
	Edit > Delete Object > Delete Object	\$delete_design_object() \$delete_object()
	Edit > Delete Object > Property	\$delete_object_property()
	Edit > Move To	\$move_design_object() \$move_object()
	File > New > Container	\$add_container()
	File > New > Directory	\$add_directory()
	File > Add Property	\$add_object_property()
	File > Add Reference	\$add_reference()
	File > Release Design	\$release_object()
	File > Explore > Contents	\$explore_contents()
	File > Explore > Go To	\$goto_directory()
	File > Explore > Parent	\$explore_parent()
	File > Explore > References	\$explore_references()
	File > Open	\$open_object()
	File > Revert Version	\$revert_version()
	File > Salvage Object	\$salvage_object()
	File > Select > All	\$select_all()
	File > Select > By Name	\$select_by_name()
	File > Select > By Type	\$select_by_type()
	File > Unselect > All	\$unselect_all()
	File > Unselect > By Name	\$unselect_by_name()

Table D-1. Pulldown Menu Paths Mapped to Functions (cont.)

Window	Pulldown Menu Path	Function
	File > Unselect > By Type	\$unselect_by_type()
	MGC > Location Map > Change Entry	\$change_location_map_entry()
	MGC > Location Map > Read Map	\$read_map()
	MGC > Location Map > Set Working Directory	\$set_working_directory()
	MGC > Location Map > Show Location Map	\$show_location_map()
	MGC > Setup Setup > Preferences	\$setup_session_defaults()
	Open	\$open_object() \$open_tool()
	Report > Show References	\$show_references()
	Report > Object Info	\$report_object_info()
	Report > Show Versions	\$show_versions()
	View > Update Window	\$update_window()
	View > View by Icon	\$view_by_icon()
	View > View by Name	\$view_by_name()
Reference	Add > Reference Property	\$add_reference_property()
	Edit > Change Reference Property	\$change_reference_property()
	Edit > Change State	\$change_reference_state()
	Edit > Delete Reference	\$delete_reference()
	Edit > Delete Reference Property	\$delete_reference_property()
	Report > Reference Info	\$report_reference_info()
Toolboxes	Edit > Add Toolbox	\$add_toolbox()
	Edit > Remove Toolbox	\$remove_toolbox()
	Edit > Save	\$save_toolbox_search_path()
	View > Tools	\$view_tools()
Tools	Report > Tool Info	\$report_tool_info()
Trash or Navigator	Edit > Empty Trash	\$empty_trash()
Trash	Edit > Search	\$search()

Table D-1. Pulldown Menu Paths Mapped to Functions (cont.)

Window	Pulldown Menu Path	Function
	Edit > Untrash Object	\$untrash_object()
Types	Edit > Load Registry	\$load_registry()
	Report > Type Info	\$report_type_info()
Versions	Edit > Copy	\$copy_version()
	Edit > Delete	\$delete_version()
	Edit > Freeze	\$freeze_version()
	Edit > Unfreeze	\$unfreeze_version()
	Report > Version Info	\$report_version_info()
	File > Unselect All	\$unselect_all()

Table D-2. Popup Menu Paths Mapped to Functions

Window	Popup Menu Item/Menu Path	Function
Component	Show Levels > Show 1 Level	\$descend_hierarchy_one_level()
	Show Levels > Show n Levels	\$descend_hierarchy_specify_level()
	Hide/Show Contents	\$show_component_hierarchy()
Component Hierarchy	Hide/Show Hierarchy Show Levels	\$show_component_hierarchy()
Configuration	Build	\$build_configuration()
	Add Versions	\$add_versions()
	Global Operations > Change References	\$change_configuration_references()
	Global Operations > Convert References	\$convert_configuration_references()
	Global Operations > Copy	\$copy_configuration()
	Global Operations > Delete Configuration	\$delete_configuration()
	Global Operations > Freeze	\$freeze_configuration()
	Global Operations > Lock Configuration	\$lock_configuration()
	Global Operations > Release	\$release_configuration()
	Global Operations > Unfreeze	\$unfreeze_configuration()

Table D-2. Popup Menu Paths Mapped to Functions (cont.)

Window	Popup Menu Item/Menu Path	Function
	Global Operations > Unlock	\$unlock_configuration()
	Hide Secondaries	\$hide_secondary_entries()
	Remove Entry	\$remove_configuration_entry()
	Report Configuration Info > Report Configuration Info	\$report_configuration_info()
	Report Configuration Info > Entry Info	\$report_entry_info()
	Report Configuration Info > Entry Verification	\$report_entry_verification()
	Report Configuration Info > References	\$report_configuration_references()
	Configuration > Select > All	\$select_all()
	View Containment	\$view_containment_hierarchy()
	Hide Secondaries	\$hide_secondary_entries()
	View Primaries	\$view_primary_hierarchy()
	View Secondaries	\$view_secondary_entries()
Navigator	Edit > Delete Object > Delete Object	\$delete_object()
	Add > Property	\$add_object_property()
	Add > Reference	\$add_reference()
	Edit > Change > Name	\$change_object_name()
	Edit > Change > Property	\$change_object_property()
	Edit > Change > Protection	\$change_protection()
	Edit > Change > References	\$change_object_references()
	Edit > Change > Version Depth	\$change_version_depth()
	Edit > Convert References	\$convert_object_references()
	Edit > Copy To	\$copy_object()
	Edit > Delete Object > Property	\$delete_object_property()
	Edit > Move To	\$move_object()
	Explore > Contents	\$explore_contents()
	Explore > Go To	\$goto_directory()

Table D-2. Popup Menu Paths Mapped to Functions (cont.)

Window	Popup Menu Item/Menu Path	Function
	Explore > Parent	\$explore_parent()
	Explore > References	\$explore_references()
	Explore > Search	\$search()
	Filter	\$setup_filter_active()
	Open	\$open_object() \$open_tool()
	Report > Object Info	\$report_object_info()
	Revert Version	\$revert_version()
	Select > All	\$select_all()
	Select > By Name	\$select_by_name()
	Select > By Type	\$select_by_type()
	Unselect	\$unselect_object()
	Unselect All	\$unselect_all()
	Unselect > By Name	\$unselect_by_name()
	Unselect > By Type	\$unselect_by_type()
	Open > Read-Only Editor	\$open_read_only_editor()
	Update Window	\$update_window()
	View by Icon	\$view_by_icon()
	View by Name	\$view_by_name()
References	Change State	\$change_reference_state()
	Delete	\$delete_reference()
	Reference Properties > Add	\$add_reference_property()
	Reference Properties > Change	\$change_reference_property()
	Reference Properties > Delete	\$delete_reference_property()
	Report Reference Info	\$report_reference_info()
	Unselect All	\$unselect_all()
Toolboxes	Add Toolbox	\$add_toolbox()
	Remove Toolbox	\$remove_toolbox()
	Save	\$save_toolbox_search_path()
	View Tools	\$view_tools()

Table D-2. Popup Menu Paths Mapped to Functions (cont.)

Window	Popup Menu Item/Menu Path	Function
Tools	Report Tool Info	\$report_tool_info()
	Open	\$open_tool()
	Search	\$search()
	Unselect All	\$unselect_all()
	View Toolboxes	\$view_toolboxes()
Trash	Empty Trash	\$empty_trash()
	Search	\$search()
	Untrash Object	\$untrash_object()
Types	Load Registry	\$load_registry()
	Report Type Info	\$report_type_info()
Versions	Copy	\$copy_version()
	Delete	\$delete_version()
	Freeze	\$freeze_version()
	Report Version Info	\$report_version_info()
	Unfreeze	\$unfreeze_version()
	Unselect All	\$unselect_all()

Logical Key Names Mapped to Functions and Scopes

A logical key name maps to a physical key, or a combination of physical keys, that performs a specific action on a specific keyboard and workstation.

As a result, the logical key name applies to whatever workstation the Pyxis Project Manager application is running on and to whatever keyboard is being used. For example, the logical key name *MenuBar* maps to F10 on the Sun Type 4 and Type 5 keyboards and the HP Series 700 ITF and PC-101 keyboards.

[Table D-3](#) maps the logical key names specific to the Pyxis Project Manager application, to their corresponding functions and scopes.

For a mapping of Pyxis Project Manager logical key names to physical keys on specific workstations and keyboards, refer to “[Logical Key Names Mapped to Physical Keys](#)” on page [806](#).

For information about keys defined by the Common User Interface, refer to the “Logical Key Name Mappings” section in the *Pyxis Common User Interface Manual* on SupportNet.

Table D-3. Key Names Mapped to Functions & Scopes

Logical Key Name	Function	Scope
ChangeName	\$change_object_name()	dmgr_model
ChangeReferences	\$change_object_references()	dmgr_model
CheckReferences	\$check_references()	dmgr_model
CloseWindow	\$close_window()	btxt_area edit_pane
Copy	\$copy_object()	dmgr_model dmgr_version_model
	\$copy_version()	dmgr_version_model
DesignBrowser	\$browse_for_object()	dmgr_session_window
Edit	\$edit_file()	dmgr_session_window
ExploreParent	\$explore_parent()	dmgr_model
GotoDirectory	\$goto_directory()	dmgr_model
MenuBar	\$popup_menu_bar()	ui_base
Monitor	\$open_session_monitor()	dmgr_session_window
	\$show_monitor()	dm_config_window
Move	\$move_object()	dmgr_trash_area dmgr_model
Navigator	\$open_navigator()	dmgr_session_window
NavigatorDirectory	\$get_navigator_directory_hard()	dmgr_model
OpenObject	\$open_object()	dmgr_model dmgr_version_model
	\$open_tool()	dmgr_tool_area
PopWindow	\$pop_window()	area session_area
Read	\$edit_file()	dmgr_session_window
ReportInfo	\$report_configuration_info()	dm_config_window
	\$report_object_info()	dmgr_model
	\$report_reference_info()	dmgr_reference_model
	\$report_tool_info()	dmgr_tool_area
	\$report_type_info()	dmgr_type_area
	\$report_version_info()	dmgr_version_model

Table D-3. Key Names Mapped to Functions & Scopes (cont.)

Logical Key Name	Function	Scope
Select	\$select_by_name()	dm_config_window dmgr_list_area dmgr_iconic_area
	\$select_by_type()	dm_config_window dmgr_list_area dmgr_iconic_area
	\$select_config_entry()	dm_config_window
	\$select_object()	dmgr_iconic_area dmgr_list_area
SelectAll	\$select_all()	dm_config_window dm_iconic_area
ShowReferences	\$show_references()	dmgr_model
UnselectAll	\$unselect_all()	dm_config_window dm_iconic_area dm_list_area dmgr_type_area

Logical Key Names Mapped to Physical Keys

The following tables map the logical key names used when working with the Pyxis Project Manager application to their corresponding physical keys on all of the platforms and keyboards that Mentor Graphics supports.

In [Table D-4](#), all of the logical key names in each of the Pyxis Project Manager windows are mapped to their physical key names on the HP ITF and HP PC-101 keyboards.

In [Table D-5](#), all of the logical key names in each of the Pyxis Project Manager windows are mapped to their physical key names on the Sun Type 3, Type 4, and Type 5 keyboards.

For a mapping of all Common User Interface logical key names, refer to the “*Logical Key Name Mappings*” section in the *Pyxis Common User Interface Manual* on SupportNet.

Table D-4. Logical Key Names for HP Keyboards

Logical Key Name	Active Window	HP ITF and HP PC-101
MenuBar	Configuration Information Monitor Navigator Reference Session Toolbox Tools Trash Types Version	F10
DesignBrowser	Configuration Information Monitor Navigator Reference Session Toolbox Tools Trash Types Version	Ctrl-F10
OpenObject	Navigator Tools Version	F1

Table D-4. Logical Key Names for HP Keyboards (cont.)

Logical Key Name	Active Window	HP ITF and HP PC-101
Navigator	Configuration Information Monitor Navigator Reference Session Toolbox Tools Trash Types Version	Shift-F1
ReportInfo	Configuration Navigator Reference Tools Types Version	Ctrl-F1
UnselectAll	Configuration Information Monitor Navigator Reference Toolbox Tools Trash Types Version	F2
Select	Configuration Navigator	Shift-F2
SelectAll	Configuration Navigator Reference Toolbox Tools Trash Version	Ctrl-F2
GotoDirectory	Navigator	F3
ExploreParent	Navigator	Shift-F3
NavigatorDirectory	Navigator	Ctrl-F3

Table D-4. Logical Key Names for HP Keyboards (cont.)

Logical Key Name	Active Window	HP ITF and HP PC-101
PopupMenu	Configuration Information Monitor Navigator Reference Session Toolbox Tools Trash Types Version	F4
Move	Navigator	F5
Copy	Navigator Version	Shift-F5
ChangeName	Navigator	Ctrl-F5
reserved for Motif	Configuration Information Monitor Navigator Reference Session Toolbox Tools Trash Types Version	F6
ShowReferences	Navigator	F7
ChangeReferences	Navigator	Shift-F7
CheckReferences	Navigator	Ctrl-F7
Monitor	Configuration Information Navigator Reference Session Version	F8

Table D-4. Logical Key Names for HP Keyboards (cont.)

Logical Key Name	Active Window	HP ITF and HP PC-101
Read	Configuration Information Monitor Navigator Reference Session Toolbox Tools Trash Types Version	F11
Edit	Configuration Information Monitor Navigator Reference Session Toolbox Tools Trash Types Version	Shift-F11
PopWindow	Configuration Information Monitor Navigator Reference Session Toolbox Tools Trash Types Version	F12
CloseWindow	Configuration Information Monitor Navigator Reference Session Toolbox Tools Trash Types Version	Shift-F12

Table D-5. Logical Key Names for Sun Keyboards

Logical Key Name	Active Window	Keyboards	
		Type 4	Type 5
OpenObject	Navigator Tools Version	F1	F1
Navigator	Configuration Information Monitor Navigator Reference Session Toolbox Tools Trash Types Version	Shift-F1	Shift-F1
ReportInfo	Configuration Navigator Reference Tools Types Version	Ctrl-F1	Ctrl-F1
UnselectAll	Configuration Monitor Navigator Referenc Toolbox Tools Trash Types Version	F2	F2
Select	Configuration Navigator	Shift-F2	Shift-F2
SelectAll	Configuration Navigator Reference Toolbox Tools Trash Types Version	Ctrl-F2	Ctrl-F2
GotoDirectory	Navigator	F3	F3

Table D-5. Logical Key Names for Sun Keyboards (cont.)

Logical Key Name	Active Window	Keyboards	
		Type 4	Type 5
ExploreParent	Navigator	Shift-F3	Shift-F3
NavigatorDirectory	Navigator	Ctrl-F3	Ctrl-F3
PopupMenu	Configuration Information Monitor Navigator Reference Session Toolbox Tools Trash Types Version	F4	F4
Move	Navigator	F5	F5
Copy	Navigator Version	Shift-F5	Shift-F5
ChangeName	Navigator	Ctrl-F5	Ctrl-F5
reserved for Motif	Configuration Information Monitor Navigator Reference Session Toolbox Tools Trash Types Version	F6	F6
ShowReferences	Navigator	F7	F7
ChangeReferences	Navigator	Shift-F7	Shift-F7
CheckReferences	Navigator	Ctrl-F7	Ctrl-F7
Monitor	Configuration Information Navigator Referenc Session Types Version	F8	F8

Table D-5. Logical Key Names for Sun Keyboards (cont.)

Logical Key Name	Active Window	Keyboards	
		Type 4	Type 5
MenuBar	Configuration Information Monitor Navigator Reference Session Toolbox Tools Trash Types Version	F10	F10
DesignBrowser	Configuration Information Monitor Navigator Reference Session Toolbox Tools Trash Types Version	Ctrl-F10	Ctrl-F10
Read	Configuration Information Monitor Navigator Reference Session Toolbox Tools Trash Types Version	F11	F11

Table D-5. Logical Key Names for Sun Keyboards (cont.)

Logical Key Name	Active Window	Keyboards	
		Type 4	Type 5
Edit	Configuration Information Monitor Navigator Reference Session Toolbox Tools Trash Types Version	Shift-F11	Shift-F11
PopWindow	Configuration Information Monitor Navigator Reference Session Toolbox Tools Trash Types Version	F12	F12
CloseWindow	Configuration Information Monitor Navigator Reference Session Toolbox Tools Trash Types Version	Shift-F12	Shift-F12

Third-Party Information

Open source and third-party software may be included in the Pyxis Products.

For third-party information, refer to *[Third-Party Software for Pyxis Products](#)*.



End-User License Agreement

The latest version of the End-User License Agreement is available on-line at:
www.mentor.com/eula

IMPORTANT INFORMATION

USE OF ALL SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE PRODUCTS. USE OF SOFTWARE INDICATES CUSTOMER'S COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.

END-USER LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Software (as defined in Section 2) and hardware (collectively "Products") between the company acquiring the Products ("Customer"), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity ("Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, in the case of Software received electronically, certify destruction of Software and all accompanying items within five days after receipt of Software and receive a full refund of any license fee paid.

1. ORDERS, FEES AND PAYMENT.

- 1.1. To the extent Customer (or if agreed by Mentor Graphics, Customer's appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement (each an "Order"), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement, any applicable addenda and the applicable quotation, whether or not those documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order or presented in any electronic portal or automated order management system, whether or not required to be electronically accepted, will not be effective unless agreed in writing and physically signed by an authorized representative of Customer and Mentor Graphics.
- 1.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will state separately in the applicable invoice. Unless timely provided with a valid certificate of exemption or other evidence that items are not taxable, Mentor Graphics will invoice Customer for all applicable taxes including, but not limited to, VAT, GST, sales tax, consumption tax and service tax. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer's sole responsibility. If Customer appoints a third party to place purchase orders and/or make payments on Customer's behalf, Customer shall be liable for payment under Orders placed by such third party in the event of default.
- 1.3. All Products are delivered FCA factory (Incoterms 2010), freight prepaid and invoiced to Customer, except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics retains a security interest in all Products delivered under this Agreement, to secure payment of the purchase price of such Products, and Customer agrees to sign any documents that Mentor Graphics determines to be necessary or convenient for use in filing or perfecting such security interest. Mentor Graphics' delivery of Software by electronic means is subject to Customer's provision of both a primary and an alternate e-mail address.

2. **GRANT OF LICENSE.** The software installed, downloaded, or otherwise acquired by Customer under this Agreement, including any updates, modifications, revisions, copies, documentation and design data ("Software") are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software solely: (a) in machine-readable, object-code form (except as provided in Subsection 5.2); (b) for Customer's internal business purposes; (c) for the term of the license; and (d) on the computer hardware and at the site authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Customer may have Software temporarily used by an employee for telecommuting purposes from locations other than a Customer office, such as the employee's residence, an airport or hotel, provided that such employee's primary place of employment is the site where the Software is authorized for use. Mentor Graphics' standard policies and programs, which vary depending on Software, license fees paid or services purchased, apply to the following: (a) relocation of Software; (b) use of Software, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be technically implemented through the use of authorization codes or similar devices); and (c) support services provided, including eligibility to receive telephone support, updates, modifications, and revisions. For the avoidance of doubt, if Customer provides any feedback or requests any change or enhancement to Products, whether in the course of receiving support or consulting services, evaluating Products, performing beta testing or otherwise, any inventions, product improvements, modifications or developments made by Mentor Graphics (at Mentor Graphics' sole discretion) will be the exclusive property of Mentor Graphics.
3. **ESC SOFTWARE.** If Customer purchases a license to use development or prototyping tools of Mentor Graphics' Embedded Software Channel ("ESC"), Mentor Graphics grants to Customer a nontransferable, nonexclusive license to reproduce and distribute executable

files created using ESC compilers, including the ESC run-time libraries distributed with ESC C and C++ compiler Software that are linked into a composite program as an integral part of Customer's compiled computer program, provided that Customer distributes these files only in conjunction with Customer's compiled computer program. Mentor Graphics does NOT grant Customer any right to duplicate, incorporate or embed copies of Mentor Graphics' real-time operating systems or other embedded software products into Customer's products or applications without first signing or otherwise agreeing to a separate agreement with Mentor Graphics for such purpose.

4. **BETA CODE.**

- 4.1. Portions or all of certain Software may contain code for experimental testing and evaluation (which may be either alpha or beta, collectively "Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. Mentor Graphics may choose, at its sole discretion, not to release Beta Code commercially in any form.
- 4.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer's use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer's evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.
- 4.3. Customer agrees to maintain Beta Code in confidence and shall restrict access to the Beta Code, including the methods and concepts utilized therein, solely to those employees and Customer location(s) authorized by Mentor Graphics to perform beta testing. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer's feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 4.3 shall survive termination of this Agreement.

5. **RESTRICTIONS ON USE.**

- 5.1. Customer may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Customer shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. Customer shall not make Products available in any form to any person other than Customer's employees and on-site contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Products and ensure that any person permitted access does not disclose or use Products except as permitted by this Agreement. Customer shall give Mentor Graphics written notice of any unauthorized disclosure or use of the Products as soon as Customer becomes aware of such unauthorized disclosure or use. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive any source code from Software. Log files, data files, rule files and script files generated by or for the Software (collectively "Files"), including without limitation files containing Standard Verification Rule Format ("SVRF") and Tcl Verification Format ("TVF") which are Mentor Graphics' trade secret and proprietary syntaxes for expressing process rules, constitute or include confidential information of Mentor Graphics. Customer may share Files with third parties, excluding Mentor Graphics competitors, provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Customer may use Files containing SVRF or TVF only with Mentor Graphics products. Under no circumstances shall Customer use Products or Files or allow their use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Products, or disclose to any third party the results of, or information pertaining to, any benchmark.
- 5.2. If any Software or portions thereof are provided in source code form, Customer will use the source code only to correct software errors and enhance or modify the Software for the authorized use. Customer shall not disclose or permit disclosure of source code, in whole or in part, including any of its methods or concepts, to anyone except Customer's employees or on-site contractors, excluding Mentor Graphics competitors, with a need to know. Customer shall not copy or compile source code in any manner except to support this authorized use.
- 5.3. Customer may not assign this Agreement or the rights and duties under it, or relocate, sublicense, or otherwise transfer the Products, whether by operation of law or otherwise ("Attempted Transfer"), without Mentor Graphics' prior written consent and payment of Mentor Graphics' then-current applicable relocation and/or transfer fees. Any Attempted Transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics' option, result in the immediate termination of the Agreement and/or the licenses granted under this Agreement. The terms of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer's permitted successors in interest and assigns.
- 5.4. The provisions of this Section 5 shall survive the termination of this Agreement.

6. **SUPPORT SERVICES.** To the extent Customer purchases support services, Mentor Graphics will provide Customer with updates and technical support for the Products, at the Customer site(s) for which support is purchased, in accordance with Mentor Graphics' then current End-User Support Terms located at <http://supportnet.mentor.com/supportterms>.

7. **LIMITED WARRANTY.**

- 7.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Products, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Products will meet Customer's requirements or that operation of Products will be uninterrupted or error free. The

warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Software under an Order and does not renew or reset, for example, with the delivery of (a) Software updates or (b) authorization codes or alternate Software under a transaction involving Software re-mix. This warranty shall not be valid if Products have been subject to misuse, unauthorized modification, improper installation or Customer is not in compliance with this Agreement. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF THE PRODUCTS TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF THE PRODUCTS THAT DO NOT MEET THIS LIMITED WARRANTY. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) PRODUCTS PROVIDED AT NO CHARGE; OR (C) BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."

7.2. THE WARRANTIES SET FORTH IN THIS SECTION 7 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO PRODUCTS PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

8. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL MENTOR GRAPHICS' OR ITS LICENSORS' LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT RECEIVED FROM CUSTOMER FOR THE HARDWARE, SOFTWARE LICENSE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 8 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

9. **HAZARDOUS APPLICATIONS.** CUSTOMER ACKNOWLEDGES IT IS SOLELY RESPONSIBLE FOR TESTING ITS PRODUCTS USED IN APPLICATIONS WHERE THE FAILURE OR INACCURACY OF ITS PRODUCTS MIGHT RESULT IN DEATH OR PERSONAL INJURY ("HAZARDOUS APPLICATIONS"). EXCEPT TO THE EXTENT THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF MENTOR GRAPHICS PRODUCTS IN OR FOR HAZARDOUS APPLICATIONS. THE PROVISIONS OF THIS SECTION 9 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

10. **INDEMNIFICATION.** CUSTOMER AGREES TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE OR LIABILITY, INCLUDING ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH THE USE OF MENTOR GRAPHICS PRODUCTS IN OR FOR HAZARDOUS APPLICATIONS. THE PROVISIONS OF THIS SECTION 10 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

11. INFRINGEMENT.

11.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Product acquired by Customer hereunder infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay costs and damages finally awarded against Customer that are attributable to such action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

11.2. If a claim is made under Subsection 11.1 Mentor Graphics may, at its option and expense: (a) replace or modify the Product so that it becomes noninfringing; (b) procure for Customer the right to continue using the Product; or (c) require the return of the Product and refund to Customer any purchase price or license fee paid, less a reasonable allowance for use.

11.3. Mentor Graphics has no liability to Customer if the action is based upon: (a) the combination of Software or hardware with any product not furnished by Mentor Graphics; (b) the modification of the Product other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of the Product as part of an infringing process; (e) a product that Customer makes, uses, or sells; (f) any Beta Code or Product provided at no charge; (g) any software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; or (h) infringement by Customer that is deemed willful. In the case of (h), Customer shall reimburse Mentor Graphics for its reasonable attorney fees and other costs related to the action.

11.4. THIS SECTION 11 IS SUBJECT TO SECTION 8 ABOVE AND STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS, AND CUSTOMER'S SOLE AND EXCLUSIVE REMEDY, FOR DEFENSE, SETTLEMENT AND DAMAGES, WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY PRODUCT PROVIDED UNDER THIS AGREEMENT.

12. TERMINATION AND EFFECT OF TERMINATION.

12.1. If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized term. Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement immediately upon written notice if Customer: (a) exceeds the scope of the license or otherwise fails to comply with the licensing or confidentiality provisions of this Agreement, or (b) becomes insolvent, files a bankruptcy petition, institutes proceedings for liquidation or winding up or enters into an agreement to assign its assets for the benefit of creditors. For any other material breach of any

provision of this Agreement, Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement upon 30 days written notice if Customer fails to cure the breach within the 30 day notice period. Termination of this Agreement or any license granted hereunder will not affect Customer's obligation to pay for Products shipped or licenses granted prior to the termination, which amounts shall be payable immediately upon the date of termination.

- 12.2. Upon termination of this Agreement, the rights and obligations of the parties shall cease except as expressly set forth in this Agreement. Upon termination, Customer shall ensure that all use of the affected Products ceases, and shall return hardware and either return to Mentor Graphics or destroy Software in Customer's possession, including all copies and documentation, and certify in writing to Mentor Graphics within ten business days of the termination date that Customer no longer possesses any of the affected Products or copies of Software in any form.
13. **EXPORT.** The Products provided hereunder are subject to regulation by local laws and United States ("U.S.") government agencies, which prohibit export, re-export or diversion of certain products, information about the products, and direct or indirect products thereof, to certain countries and certain persons. Customer agrees that it will not export or re-export Products in any manner without first obtaining all necessary approval from appropriate local and U.S. government agencies. If Customer wishes to disclose any information to Mentor Graphics that is subject to any U.S. or other applicable export restrictions, including without limitation the U.S. International Traffic in Arms Regulations (ITAR) or special controls under the Export Administration Regulations (EAR), Customer will notify Mentor Graphics personnel, in advance of each instance of disclosure, that such information is subject to such export restrictions.
14. **U.S. GOVERNMENT LICENSE RIGHTS.** Software was developed entirely at private expense. The parties agree that all Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to U.S. FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. government or a U.S. government subcontractor is subject solely to the terms and conditions set forth in this Agreement, which shall supersede any conflicting terms or conditions in any government order document, except for provisions which are contrary to applicable mandatory federal laws.
15. **THIRD PARTY BENEFICIARY.** Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, Microsoft Corporation and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.
16. **REVIEW OF LICENSE USAGE.** Customer will monitor the access to and use of Software. With prior written notice and during Customer's normal business hours, Mentor Graphics may engage an internationally recognized accounting firm to review Customer's software monitoring system and records deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FlexNet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. The provisions of this Section 16 shall survive the termination of this Agreement.
17. **CONTROLLING LAW, JURISDICTION AND DISPUTE RESOLUTION.** The owners of certain Mentor Graphics intellectual property licensed under this Agreement are located in Ireland and the U.S. To promote consistency around the world, disputes shall be resolved as follows: excluding conflict of laws rules, this Agreement shall be governed by and construed under the laws of the State of Oregon, U.S., if Customer is located in North or South America, and the laws of Ireland if Customer is located outside of North or South America. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of the courts of Portland, Oregon when the laws of Oregon apply, or Dublin, Ireland when the laws of Ireland apply. Notwithstanding the foregoing, all disputes in Asia arising out of or in relation to this Agreement shall be resolved by arbitration in Singapore before a single arbitrator to be appointed by the chairman of the Singapore International Arbitration Centre ("SIAC") to be conducted in the English language, in accordance with the Arbitration Rules of the SIAC in effect at the time of the dispute, which rules are deemed to be incorporated by reference in this section. Nothing in this section shall restrict Mentor Graphics' right to bring an action (including for example a motion for injunctive relief) against Customer in the jurisdiction where Customer's place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.
18. **SEVERABILITY.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.
19. **MISCELLANEOUS.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements. Some Software may contain code distributed under a third party license agreement that may provide additional rights to Customer. Please see the applicable Software documentation for details. This Agreement may only be modified in writing, signed by an authorized representative of each party. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.