ENOVIA Synchronicity DesignSync Data Manager

V6R2013

DFII SKILL Programming Interface Guide

Copyrights and Trademarks

NOTE: This manual was generated directly from the online help with minimal reformatting. The PDF version is optimized for printing and does not contain active cross-reference links or animated use cases. Because the intent of the help is to be an online guide, there may be shortcomings in its organization and general usability as a printed document. The PDF version was created before limited changes were made to WebHelp. For the most current information, see the product's online help.

Dassault Systèmes ENOVIA

175 Wyman Street,

Waltham, MA 02451

Telephone 781.810.3500

Email: enovia.info@3ds.com

http://www.3ds.com

Table Of Contents

Release Information 1

Documentation 1

Selecting the appropriate release 1

Available Release-Specific Documentation 1

Locating the Release Specific Documentation 1

Product Enhancement Overview 2

General and Open Issues 2

Closed Issues 2

Installation 2

Introduction 3

Syntax Conventions 3

Error Handling and Diagnostics 5

Error Handling in Function Invocations 5

Setting a Trace 5

Return Values 5

Return Values and Background Commands 5

Revision Control Functions 7

dssAddFileP 7

dssBranchCellP 7

dssBranchCellViewP 8

dssBranchLibraryP 9

dssCancelCellViewP 9

dssCancelFileP... 11

dssCheckinCategoryP 12

dssCheckinCellP.. 14

dssCheckinCellViewP.. 16

dssCheckinFileP.. 18

dssCheckinHierarchyP 21

dssCheckinLibraryP... 25

dssCheckoutCategoryP... 27

dssCheckoutCellP 29

dssCheckoutCellViewP.. 31

dssCheckoutFileP.. 33

dssCheckoutHierarchyP 36

dssCheckoutLibraryP.. 40

dssCompareViewsP 42

dssCompareViewsHandlerP.. 45

dssCompareViewsListHandlersP... 48

dssCompareViewsRemoveHandlerP.. 49

dssConfigureLibraryP 49

dssCreateCellViewP.. 50

dssDeleteCategoryP.. 51

dssDeleteCellP 53

dssDeleteCellViewP 55

dssDeleteFileP 57

dssDeleteLibraryP 59

dssDeleteTemporaryViewsP 61

dssDeleteVersionP 62

dssFetchCellViewVersionP........ 62

dssFetchLockedP 63

dssGetFileTagsP 66

dssGetFileVersionP 67

dssGetFileVersionsP 69

dssGetTagListP 70

dssGetViewPathP........ 72

dssGetViewTagsP 72

dssGetViewVersionP 73

dssGetViewVersionsP 75

dssIsFileLockedP........ 76

dssIsViewLockedP........ 77

dssJoinLibraryP 78

dssLibraryStatusP........ 80

dssListHierarchyP........ 82

dssTagCategoryP 85

dssTagCellP 86

dssTagCellViewP........ 87

dssTagFileP........ 89

dssTagHierarchyP 91

dssTagLibraryP.. 94

dssUnlockCellViewP.. 95

dssUnlockFileP.. 96

dssViewDataSheetP.. 98

dssViewVersionHistoryP.. 99

Menu Customization Functions 103

Customizing the Synchronicity Menu.. 103

dssMenuAddItemP 103

dssMenuAddValidItemP 104

dssMenuListItemsP 106

dssMenuListMenuP 106

dssMenuLoadConfigP 107

dssMenuRefreshP 107

dssMenuRemoveItemAllP 108

dssMenuRemoveItemP 108

dssMenuRemoveValidItemP 109

dssMenuSaveConfigP 110

dssRefreshWindowBannerP.. 110

Miscellaneous Functions... 113

dssChangeDefaultsContextP... 113

dssChangeUserLevelP 114

dssEnableDebugP... 114

dssExecuteTclP.. 115

dssHelpP 116
dssSetWorkspaceRootPathP 116
dssGetWorkspaceRootPathP 117
Getting Assistance 119
Using Help 119
Getting a Printable Version of Help 120
Accessing Product Documentation 120
Contacting ENOVIA 121
DesignSync Glossary 123
A 123
B 124
C 125
D 129
E 131
F 132
G 133
H 133
I 134
J 135
K 135
L 136
M 138
N 142

O 142
P 143
Q 145
R 145
S 149
T 152
U 154
V 155
W 156
X 157
Y 157
Z 157
Index 159

Release Information

Documentation

Release-specific information is located on the Dassault Systèmes support website in the Program Directory (http://media.3ds.com/support/progdir/). The Program Directory contains release-specific information for all major DesignSync releases beginning with V6R2009x.

Selecting the appropriate release

1. Open the Program Directory (http://media.3ds.com/support/progdir/). You may be required to enter your username and password to access information on the 3ds support site.
2. Select the following options in the top bar:

   Select Line: **Version 6**
   Select Level: **V6R2013**
   Select Sub-Level: (use default)

   Note: By default, the sub-level is always the most current version of the Program Directory files for the selected Level. There should never be a reason that information you need for a release is not in the most current version.

Available Release-Specific Documentation

The documents listed in the following table are available.

Product Enhancement Overview	Contains the list of new features and enhancements for the release.
General and Open Issues	Contains any known release issues, platform support information, platform configuration information, and system configuration recommendations for the release.
Closed Issues	Contains a complete list of closed issues for the release.
Installation	Installation instructions for DesignSync clients on all supported platforms. For server configuration information, see the *ENOVIA Synchronicity DesignSync Administrator's Guide*.

Locating the Release Specific Documentation

Product Enhancement Overview

1. In the left frame, select **ENOVIA** in the **Product Enhancement Overview** Section. This opens the Product Enhancement Overview index in the right frame.
2. Navigate to the IP Work-in-Progress section and select **Synchronicity DesignSync Data Manager**, or use your browser search functionality to search for **Synchronicity DesignSync Data Manager**. Selecting **Synchronicity DesignSync Data Manager** opens the Product Enhancement Overview for DesignSync.

General and Open Issues

1. In the left frame, select **ENOVIA** in the **General and Open Issues** Section
2. Navigate to the IP Work-in-Progress: Semiconductor EDA section and select **Synchronicity DesignSync Data Manager (SYN)**, or use your browser search functionality to search for **Synchronicity DesignSync Data Manager**. Selecting **Synchronicity DesignSync Data Manager (SYN)** opens the General and Open Issues for DesignSync.

Closed Issues

1. In the left frame, select **List of Closed Issues** in the **Closed Issues** Section.
2. Use your browser search functionality to search for Synchronicity. This will bring you to the section of the closed issues list that includes the following products:

Synchronicity DesignSync (including DSclipse, and DSVS plug-ins)
Synchronicity DesignSync Add-On for DFII
Synchronicity DesignSync Add-On for DSMW
Synchronicity DesignSync Add-On for DSCD
Synchronicity DesignSync Add-On for CTS
Synchronicity ProjectSync

Note: Not all releases include closed issues for all DesignSync products.

Installation

1. In the left frame, select **ENOVIA Server** in the **Installation** Section
2. Select **ENOVIA Synchronicity DesignSync Data Manager** in the navigation links at the top of the page or use your browser search functionality to search for **ENOVIA Synchronicity DesignSync Data Manager**.
3. Select the **Installing Synchronicity DesignSync Data Manager** link to open the Installation document.

Introduction

ENOVIA Synchronicity DesignSync® Data Manager DFII(TM) is the integration of many DesignSync® design-management (DM) capabilities into the Cadence Design Systems DFII environment.

The **DesignSync Data Manager DFII SKILL Programming Interface Guide** contains function descriptions for the DesignSync DFII SKILL ™ API functions available in the DesignSync DFII environment. For general instruction on using DesignSync DFII, see the DesignSync Data Manager DFII User's Guide. For a description of SKILL variables you can set to customize your environment, see the DesignSync Data Manager DFII User's Guide: Using SKILL Variables.

Syntax Conventions

The following syntax conventions are used in the SKILL syntax descriptions shown in this document:

- `z_argument`: Words with a prefix containing one or more characters followed by an underscore indicate arguments for which you must substitute a name or a value. The characters before the underscore (_) in the word indicate the data type that this argument can take. These data types include the following:

  `t`: text; a string

  `l`: list

  `S`: symbol or character string

  `x`: integer

  `g`: general; usually a boolean (t/nil) value unless the description indicates otherwise. Note that an argument with the `g_` prefix can take any value, as in SKILL, `nil` represents false and any other value represents true.

  `r`: defstruct, a named structure that is a collection of one or more variables.

  Arguments that accept more than one data type use a combination of the characters above. For example, if an argument accepts a list or a text string, the argument has the `tl_` prefix.

- `?argument`: Words with a ? prefix are **keyed** arguments. If an argument is a keyed argument, you include the key as well as the value for the key in the function invocation as in the following example:

```
dssCheckinCellP("rtllib" "top" ?force t)
```

All keyed arguments are optional. Note that key names are case sensitive.

- **|** Vertical bars separate possible choices for a single argument. They take precedence over other characters.
- [ ] Brackets denote optional arguments. When used with vertical bars, they enclose a list of choices from which you can choose one.
- ... Three dots (...) indicate that you can repeat the previous argument. If they are used with brackets, you can specify zero or more arguments. If they are used without brackets, you need to specify at least one argument, but you can specify more.

argument...	Specify at least one, but more are possible.
[argument]...	Specify zero or more.

- => A right arrow precedes the possible values that can be returned by a SKILL function. It is represented with an equal sign and a greater than sign (=>).
- / A slash separates the possible values that can be returned by a SKILL function. **Note**: If no value is specified, the return value is undefined.

For more details about SKILL syntax, see the Cadence SKILL documentation.

Error Handling and Diagnostics

Error Handling in Function Invocations

When a function is called, the SKILL interpreter checks its arguments, ensuring the correct number and data types of arguments. If an argument's data type is inappropriate, the SKILL interpreter raises an error. Note that an argument with the `g_` prefix can take any value; as in SKILL, `nil` represents false and any other value represents true. The SKILL interpreter also raises errors for invalid arguments, such as an argument that specifies a nonexistent library.

If you need to handle these errors in a specific way, use the Cadence `errset` function to trap errors returned by the DesignSync DFII SKILL functions. See the Cadence SKILL documentation for a description of the `errset` function.

Setting a Trace

To create a trace of the DesignSync (stclc) session that runs beneath the DesignSync DFII session, you can invoke the DesignSync `synctrace` command. To do so, call the `synctrace` command from within the `dssExecuteTclP` function as follows:

```
dssExecuteTclP("synctrace set 0")
```

To turn off tracing:

```
dssExecuteTclP("synctrace unset 0")
```

See the `synctrace` command description in the **ENOVIA Synchronicity Command Reference** for details.

Return Values

DesignSync DFII commands return a list of pass and fail counts. In general, the first integer represents the number of objects processed successfully and the second integer represents the number of failures. For commands that do not operate on a set of objects, a single return value of `t` or `nil` is returned, indicating the success or failure of the operation.

Return Values and Background Commands

Background commands do not necessarily run immediately and thus do not provide meaningful return values immediately. DesignSync DFII adds new background commands to the Background Queue. If you run DesignSync DFII commands in the background using the `?background` option, the return value is `(0,0)`in the cases where a pass/fail count is normally returned, and `t` in the cases where a `t/nil` value is normally returned. The t value indicates that the DesignSync DFII successfully added the command to the Background Queue.

Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. In the Background Queue, DesignSync DFII SKILL commands are differentiated from graphical interface commands by the characters `(API)`, which follow the command entry. See the DesignSync Data Manager DFII User's Guide: Displaying the Background Queue for more information.

When the background SKILL command runs, DesignSync DFII outputs status messages to the standard output window, for example, to the Command Interface Window (CIW) if you invoked the background command in the CIW. If you run a command in the background using a SKILL command with the `?background` option rather than through the graphical interface, DesignSync DFII does not display a pop-up window when the command completes even if you have set the **Display pop-up windows when background operations complete** option or the corresponding SKILL variable: `syncDisplayBackgroundCompletePopup`.

Revision Control Functions

dssAddFileP

```
dssAddFileP(t_moduleName tl_fileNames [?silent g_silent])

=> nil/(x_pass x_fail)
```

Description

Adds specified objects to the specified module.

Arguments

`t_moduleName`	The workspace address of the module. This can be a simple module name, module instance name or full workspace address. For more information on referring to modules, see Specifying Module Objects for Operations in the DesignSync help.
`t_fileNames`	One or more file object(s) to be added to the module. You can add objects with wildcard characters.
`g_silent`	Run silently (`t`). (Default) Command reports output (`nil`).

Value Returned

Function returns a count of the number of objects successfully added, and the number which failed, unless there is an error processing the arguments.

dssBranchCellP

```
dssBranchCellP(
  t_libName t_cellName t_branchName
  [?checkLocked g_checkLocked]
  [?setSelector g_setSelector]
  [?silent g_silent]
)
=> t/nil
```

Description

Creates a branch for the specified cell.

Note: You cannot specify a module to branch.

Arguments

`t_libName`	Library name. (Required)
`t_cellName`	Cell name. (Required)
`t_branchName`	Branch name. (Required)
`g_checkLocked`	Discontinue branching if there are checked out or new objects (`t`). When set to (t), if any checked out or new objects are found, then no objects will be branched. By default, branching continues even if there are checked out or new objects (`nil`).
`g_setSelector`	Set the persistent selector list to match the branch (`t`). By default, the persistent selector list is not updated, so revision-control operations continue on the branch associated with the cell prior to creating the new branch (`nil`).
`g_silent`	Run silently (`t`). (Default)

Value Returned

Returns `t` if the branch has been created successfully; otherwise, returns `nil`.

dssBranchCellViewP

```
dssBranchCellViewP(
  t_libName t_cellName t_viewName t_branchName
  [?checkLocked g_checkLocked] [?silent g_silent]
)
=> t/nil
```

Description

Creates a branch for the specified cell view.

Note: You cannot specify a module to branch.

Arguments

`t_libName`	Library name. (Required)
`t_cellName`	Cell name. (Required)
`t_viewName`	View name. (Required)
`t_branchName`	Branch name. (Required)
`g_checkLocked`	Discontinue branching if there are checked out or new objects (`t`). When set to (t), if any checked out or new objects are found, then no objects will be branched. By default, branching continues even if there are checked out or new objects (`nil`).
`g_silent`	Run silently (`t`). (Default)

Value Returned

Returns `t` if the branch has been created successfully; otherwise, returns `nil`.

dssBranchLibraryP

```
dssBranchLibraryP(
  t_libName t_branchName [?checkLocked g_checkLocked]
  [?setSelector g_setSelector] [?silent g_silent]
)
=> t/nil
```

Description

Creates a branch for the specified library.

Note: You cannot specify a module to branch.

Arguments

`t_libName`	Library name. (Required)
`t_branchName`	Branch name. (Required)
`g_checkLocked`	Discontinue branching if there are checked out or new objects (`t`). When set to (t), if any checked out or new objects are found, then no objects will be branched. By default, branching continues even if there are checked out or new objects (`nil`).
`g_setSelector`	Set the persistent selector list to match the branch (`t`). By default, the persistent selector list is not updated, so revision-control operations continue on the branch associated with the workspace prior to creating the new branch (`nil`).
`g_silent`	Run silently (`t`). (Default)

Value Returned

Returns `t` if the branch has been created successfully; otherwise, returns `nil`.

dssCancelCellViewP

```
dssCancelCellViewP(
  t_libName t_cellName t_viewName
  [?mode t_mode] [?force g_force]
  [?retain g_retain] [?silent g_silent]
  [?background g_background]
)
=> nil/(x_pass x_fail)
```

Description

Cancels the checkout of a single cell view.

Arguments

`t_libName`	Library name. (Required)
`t_cellName`	Cell name. (Required)
`t_viewName`	View name. (Required)
`t_mode`	Fetch mode ("`keep`", "`share`", or "`mirror`"). By default, the mode matches the default fetch mode. If the default fetch mode is "`lock`", the `dssCancelCellViewP` default for `t_mode` is "`keep`". See the DesignSync Data Manager DFII User's Guide:Selecting a Default Fetch Mode to learn how to set the default fetch mode.
`g_force`	Overwrite the cell view even if it is locally modified (`t`). If a cell view is locally modified and you set `t_mode` to "`share`" or "`mirror`", but you do not set `g_force` to `t`, the cancel operation fails.
`g_retain`	Retain the "last modified" timestamps of the objects that remain in your workspace (`t`), or make the timestamps the time of the cancel operation (`nil`). The default is `nil` unless defined otherwise from SyncAdmin (see SyncAdmin Help). The retain option is only meaningful when leaving physical copies of objects in your workspace (`keep` mode), and is silently ignored otherwise.
`g_silent`	Run silently (`t`). (Default)
`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.

Value Returned

Returns a list of pass and fail counts; the first integer represents the number of checkouts successfully canceled and the second integer represents the number of failures. The `dssCancelCellViewP` function lets you cancel a single checkout, so the returned list is (1 0) if the cancel is successful and (0 1) if the cancel fails. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssCancelFileP

```
dssCancelFileP(
  t_fileName [?mode t_mode] [?force g_force]
  [?retain g_retain] [?silent g_silent]
  [?background g_background]
)
=> nil/(x_pass x_fail)
```

Description

Cancels the checkout of a single file.

You can specify an absolute or relative filename. Filenames can be relative to the current working directory or to any library on the library path. For example, if library `acc` is on your library path, then you can specify the `cdsinfo.tag` file for that library as `acc/cdsinfo.tag`, even though the `acc` library directory might be anywhere on disk. If a library name exists, and there is also a directory within the current working directory of the same name, the library name is used.

Arguments

`t_fileName`	A filename. (Required) A filename can be absolute or relative to the current working directory or to any library on the library path. **Note**: You must specify a filename; other file objects that resolve to directories, libraries, cells, and views are not supported by the `dssCancelFileP` function. Likewise, you cannot specify the type of view object that DesignSync creates, for example: `~/ttlLib/and2/symbol.sync.cds`. These objects are not actual files; thus, you cannot apply the `dssCancelFileP` function to this type of object.
`t_mode`	Fetch mode ("`keep`", "`share`", or "`mirror`"). By default, the mode matches the default fetch mode. If the default fetch mode is "`lock`", the `dssCancelFileP` default for `t_mode` is "`keep`". See the DesignSync Data Manager DFII User's Guide:Selecting a Default Fetch Mode to learn how to set the default fetch mode.
`g_force`	Overwrite the file even if it is locally modified (`t`). If a file is

	locally modified and you set `t_mode` to "`share`" or "`mirror`", but you do not set `g_force` to `t`, the cancel operation fails.
`g_retain`	Retain the "last modified" timestamps of the objects that remain in your workspace (`t`), or make the timestamps the time of the cancel operation (`nil`). The default is `nil` unless defined otherwise from SyncAdmin (see SyncAdmin Help). The retain option is only meaningful when leaving physical copies of objects in your workspace (`keep` mode), and is silently ignored otherwise.
`g_silent`	Run silently (`t`). (Default)
`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.

Value Returned

Returns a list of pass and fail counts; the first integer represents the number of checkouts successfully canceled and the second integer represents the number of failures. The `dssCancelFileP` function lets you cancel a single checkout, so the returned list is (1 0) if the cancel is successful and (0 1) if the cancel fails. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssCheckinCategoryP

```
dssCheckinCategoryP(
  t_libName tl_catNames [?viewNames l_viewNames]
  [?mode t_mode] [?force g_force] [?comment t_comment]
  [?skip g_skip] [?new g_new] [?nested g_nested]
  [?retain g_retain] [?silent g_silent] [?iflock g_lock]
  [?background g_background] [?branch t_branch] [?tag g_tag]
)
=> nil/(x_pass x_fail)
```

Description

Checks in objects of one or more categories. You can check in all the objects in a category at one time or specify views to check in.

Arguments

`t_libName`	Library name. (Required)
`tl_catNames`	One or more category names. (Required)
`l_viewNames`	One or more view name(s) to be checked in. (Optional). Checks in all views by default.
`t_mode`	Check-in mode ("`lock`", "`share`", "`mirror`", or "`keep`"). By default, the mode matches the default fetch mode. See the DesignSync Data Manager DFII User's Guide:Selecting a Default Fetch Mode to learn how to set the default fetch mode.
`g_force`	Force a checkin to create a new version in the vault (`t`). By default, DesignSync DFII does not force a checkin (`nil`).
`t_comment`	Check-in comment. By default, no check-in comment is supplied. However, if DesignSync DFII has been configured to require a comment of a particular length, a check-in comment is required.
`g_skip`	Skip version (`t`). By default, version skipping is not allowed (`nil`).
`g_new`	Allow new (or retired) items to be checked in (`t`). By default, checking in new items is not allowed (`nil`).
`g_nested`	Apply to nested category contents (`t`). (Default) **Note:** If `g_nested` is set to `t` but one or more nested category files are missing from your workspace, DesignSync DFII automatically fetches the missing category files and processes the specified objects.
`g_retain`	Retain the "last modified" timestamps of the objects that remain in your workspace (`t`), or make the timestamps the check-in time (`nil`). The default is `nil` unless defined otherwise from SyncAdmin (see SyncAdmin Help). The retain option is only meaningful when leaving physical copies of objects in your workspace (`lock` and `keep` modes), and is silently ignored otherwise.
`g_silent`	Run silently (`t`). (Default)
`g_iflock`	Specifies whether to check in all modified objects in the workspace (f) (default) or only targeted files (t). Targeted files include: locked DesignSync vault files or module members and added, renamed, or removed module objects.
`g_background`	Run command in the background (`t`). By default, commands

	run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.
`t_branch`	Checks the object into the specified branch, rather than checking the object into the branch it was checked out from. The branch tag can be any valid branch selector, including auto(*branchname*). **Note:** This option is not applicable to modules. If used with the -modulecontext option, or on module data, the command fails with an appropriate error.
`g_tag`	Tags the object version or module version on the server with the specified tag name. For module objects, all objects are evaluated before the checkin begins. If the module cannot be tagged, for example if the user does not have access to add a tag or because the tag exists and is immutable, the entire module checkin fails. **Note:** Individual module objects cannot have tags. Only the module itself can be tagged.

Value Returned

Returns a list of pass and fail counts; the first integer represents the number of objects successfully checked in and the second integer represents the number of failures. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssCheckinCellP

```
dssCheckinCellP(
  t_libName tl_cellNames [?viewNames l_viewNames]
  [?mode t_mode] [?force g_force] [?comment t_comment]
  [?skip g_skip] [?new g_new] [?retain g_retain]
  [?silent g_silent] [?background g_background]
  [?branch t_branch] [?iflock g_lock] [?tag g_tag]
)
=> nil/(x_pass x_fail)
```

Description

Checks in one or more cells, either all the objects in each cell or a specified set of cell views.

Arguments

`t_libName`	Library name. (Required)
`tl_cellNames`	One or more cell name(s) to be checked in. (Required)
`l_viewNames`	One or more view name(s) to be checked in. (Optional). Checks in all views by default.
`t_mode`	Check-in mode ("`lock`", "`share`", "`mirror`", or "`keep`"). By default, the mode matches the default fetch mode. See the DesignSync Data Manager DFII User's Guide:Selecting a Default Fetch Mode to learn how to set the default fetch mode.
`g_force`	Force a checkin to create new versions in the vault (`t`). By default, DesignSync DFII does not force a checkin (`nil`).
`t_comment`	Check-in comment. By default, no check-in comment is supplied. However, if DesignSync DFII has been configured to require a comment of a particular length, a check-in comment is required.
`g_skip`	Skip version (`t`). By default, version skipping is not allowed (`nil`).
`g_new`	Allow new (or retired) items to be checked in (`t`). By default, checking in new items is not allowed (`nil`).
`g_retain`	Retain the "last modified" timestamps of the objects that remain in your workspace (`t`), or make the timestamps the check-in time (`nil`). The default is `nil` unless defined otherwise from SyncAdmin (see SyncAdmin Help). The retain option is only meaningful when leaving physical copies of objects in your workspace (`lock` and `keep` modes), and is silently ignored otherwise.
`g_silent`	Run silently (`t`). (Default)
`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.
`t_branch`	Checks the object into the specified branch, rather than

	checking the object into the branch it was checked out from. The branch tag can be any valid branch selector, including auto(*branchname*). **Note:** This option is not applicable to modules. If used with the -modulecontext option, or on module data, the command fails with an appropriate error.
`g_iflock`	Specifies whether to check in all modified objects in the workspace (f) (default) or only targeted files (t). Targeted files include: locked DesignSync vault files or module members and added, renamed, or removed module objects.
`g_tag`	Tags the object version or module version on the server with the specified tag name. For module objects, all objects are evaluated before the checkin begins. If the module cannot be tagged, for example if the user does not have access to add a tag or because the tag exists and is immutable, the entire module checkin fails. **Note:** Individual module objects cannot have tags. Only the module itself can be tagged.

Value Returned

Returns a list of pass and fail counts; the first integer represents the number of objects successfully checked in and the second integer represents the number of failures. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssCheckinCellViewP

```
dssCheckinCellViewP(
  t_libName t_cellName t_viewName [?mode t_mode]
  [?force g_force] [?comment t_comment] [?skip g_skip]
  [?new g_new] [?silent g_silent] [?background g_background]
 [?branch t_branch] [?iflock g_lock] [?tag g_tag]
)
=> nil/(x_pass x_fail)
```

Description

Checks a cell view into the specified library.

Arguments

`t_libName`	Library name. (Required)
`t_cellName`	Cell name. (Required)
`t_viewName`	View name. (Required)
`t_mode`	Check-in mode (`lock` , `share`, `mirror`, or `keep`). By default, the mode matches the default fetch mode. See DesignSync DFII Help:Selecting a Default Fetch Mode to learn how to set the default fetch mode.
`g_force`	Force a checkin to create a new version in the vault (`t`). By default, DesignSync DFII does not force a checkin (`nil`).
`t_comment`	Provide a check-in comment. By default, no check-in comment is supplied. However, if DesignSync DFII has been configured to require a comment of a particular length, a check-in comment is required.
`g_skip`	Skip version (`t`). By default, version skipping is not allowed (`nil`).
`g_new`	Allow new (or retired) items to be checked in (`t`). By default, checking in new items is not allowed (`nil`).
`g_silent`	Run silently (`t`). (Default)
`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.
`t_branch`	Checks the object into the specified branch, rather than checking the object into the branch it was checked out from. The branch tag can be any valid branch selector, including auto(*branchname*). **Note:** This option is not applicable to modules. If used with the -modulecontext option, or on module data, the command fails with an appropriate error.
`g_iflock`	Specifies whether to check in all modified objects in the workspace (f) (default) or only targeted files (t). Targeted files include: locked DesignSync vault files or module members and added, renamed, or removed module objects.
`g_tag`	Tags the object version or module version on the server with the specified tag name. For module objects, all objects are evaluated before the checkin begins. If the module cannot be tagged, for example if

the user does not have access to add a tag or because the tag exists and is immutable, the entire module checkin fails.

Note: Individual module objects cannot have tags. Only the module itself can be tagged.

Value Returned

Returns a list of pass and fail counts; the first integer represents the number of objects successfully checked in and the second integer represents the number of failures. The `dssCheckinCellViewP` function lets you check-in a single cell view only, so the returned list is (1 0) if the check-in operation is successful and (0 1) if the check-in operation fails. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssCheckinFileP

```
dssCheckinFileP
  (tl_fileNames [?mode t_mode] [?force g_force]
  [?comment t_comment] [?skip g_skip] [?new g_new]
  [?retain g_retain] [?silent g_silent] [?tag g_tag]
  [?background g_background][?recursive g_recursive]
  [?branch t_branch] [?iflock g_lock]
  [?moduleContext t_moduleContext]
)
=> nil/(x_pass x_fail)
```

Description

Checks in one or more file objects.

You can specify absolute or relative filenames to be checked in. Filenames can be relative to the current working directory or to any library on the library path. For example, if library `acc` is on your library path, then you can specify the `cdsinfo.tag` file for that library as `acc/cdsinfo.tag`, even though the `acc` library directory might be anywhere on disk. If a library name exists, and there is also a directory within the current working directory of the same name, the library name is used.

Specify wildcards for filenames using glob-style expressions.

Note:

For wildcards, filenames in the current working directory take precedence over library names. That is, a glob expression of `lib*` will not match libraries `libA` and `libB` if similarly named files exist in the current working directory; the `dssCheckinFileP` function first expands regular expressions against the current directory, and then performs library matching.

Arguments

`tl_fileNames`	One or more file object(s) to be checked in. (Required) You can specify file objects as glob-style expressions. A file object can be: A filename, specified as a full path or a path relative to the current working directory. A filename, specified relative to a library, for example `<libname>/cdsinfo.tag` or `<libname>/cellname/prop.xx`. A directory name, either a full path or a path relative to the current working directory. A library name. A cell name, specified as `<libname>/<cellname>`. A view name, specified as `<libname>/<cellname>/<viewname>`. A module name. **Note**: DesignSync creates objects called <name>`.sync.cds` to represent Cadence views, where <name> corresponds to the name of the view folder, for example: `~/ttlLib/and2/symbol.sync.cds`. These objects are not actual files; thus, you cannot apply the `dssCheckinFileP` function to this type of object.
`t_mode`	Check-in mode ("`lock`", "`share`", "`mirror`", or "`keep`"). By default, the mode matches the default fetch mode. See the DesignSync Data Manager DFII User's Guide:Selecting a Default Fetch Mode to learn how to set the default fetch mode.
`g_force`	Force a checkin to create new versions in the vault (`t`). By default, DesignSync DFII does not force a checkin (`nil`).
`t_comment`	Check-in comment. By default, no check-in comment is supplied. However, if DesignSync DFII has been configured to require a comment of a particular length, a check-in comment is required.

`g_skip`	Skip version (`t`). By default, version skipping is not allowed (`nil`).
`g_new`	Allow new (or retired) items to be checked in (`t`). By default, checking in new items is not allowed (`nil`).
`g_retain`	Retain the "last modified" timestamps of the objects that remain in your workspace (`t`), or make the timestamps the check-in time (`nil`). The default is `nil` unless defined otherwise from SyncAdmin (see SyncAdmin Help). The retain option is only meaningful when leaving physical copies of objects in your workspace (`lock` and `keep` modes), and is silently ignored otherwise.
`g_silent`	Run silently (`t`). (Default)
`g_tag`	Tags the object version or module version on the server with the specified tag name. For module objects, all objects are evaluated before the checkin begins. If the module cannot be tagged, for example if the user does not have access to add a tag or because the tag exists and is immutable, the entire module checkin fails. **Note:** Individual module objects cannot have tags. Only the module itself can be tagged.
`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.
`g_recursive`	Run checkin recursively (t). (Default) **Note:** If you check in a folder, you must specify recursive to check in the contents of the folder.
`t_branch`	Checks the object into the specified branch, rather than checking the object into the branch it was checked out from. The branch tag can be any valid branch selector, including auto(*branchname*). **Note:** This option is not applicable to modules. If used with the -modulecontext option, or on module data, the command fails with an appropriate error.
`g_iflock`	Specifies whether to check in all modified objects in the

	workspace (f) (default) or only targeted files (t). Targeted files include: locked DesignSync vault files or module members and added, renamed, or removed module objects.
`t_moduleContext`	If you are checking in objects to two different modules in the same workspace, use two separate checkin operations. The module context to restrict the operation to. The module must have its base directory at, or above, the level of the library being checked in. If you do not specify a module context, the operation applies to all objects specified. **Note:** You can only specify one module.

Value Returned

Returns a list of pass and fail counts; the first integer represents the number of objects successfully checked in and the second integer represents the number of failures. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssCheckinHierarchyP

```
dssCheckinHierarchyP(
  t_libName t_cellName tl_viewNames
  [?switchUsing t_switchUsing]
  [?switchList l_switchList] [?stopList l_stopList]
  [?switchLibChoice S_switchLibChoice]
  [?switchLibNames l_switchLibNames]
  [?processViews gl_processViews]
  [?processFiles gS_processFiles]
 [?tag g_tag] [?mode t_mode] [?silent g_silent]
  [?comment t_comment] [?force g_force] [?iflock g_lock]
  [?new g_new] [?skip g_skip] [?retain g_retain]
  [?branch t_branch] [?background g_background]
)
=> nil/(x_pass x_fail)
```

Description

Checks objects into a design hierarchy. To identify the cells in a design hierarchy, DesignSync DFII scans the hierarchy, beginning with the top-level cell views you specify using the `tl_viewNames` argument. Then, DesignSync DFII descends into the views indicated by the `t_switchUsing` argument. You can use the `t_switchUsing` argument to specify that DesignSync DFII descend into one or more views you specify in a switch list (using the `l_switchList` argument). You can instead have DesignSync DFII descend into all instantiated views or all views that exist for a cell by setting the `t_switchUsing` argument to `"instantiatedView"` or `"allViews"`, respectively. Use the `l_stopList` argument to indicate at which views DesignSync DFII is to stop scanning. DesignSync DFII also offers other hierarchy controls, such as limiting which libraries are scanned using the `S_switchLibChoice` argument and limiting which views are checked in using the `g_processViews` argument.

Notes:

- For DesignSync DFII to scan the hierarchy, the cells must be in your local workspace.
- DesignSync DFII does not scan through libraries that have been filtered out using the `l_switchLibNames` argument. For example, suppose a cell in **library_1** references a cell in **library_2**, which references a cell in **library_3**. If **library_2** is filtered out in the `l_switchLibNames` argument, the cell in **library_3** is not found.

Arguments

`t_libName`	Top library name of hierarchy to be checked in. (Required)
`t_cellName`	Top cell name of hierarchy to be checked in. (Required)
`tl_viewNames`	Top-level view names of hierarchies to be checked in. (Required) Can be given a single view, a string, or a list of views.
`t_switchUsing`	Indicates how the design hierarchy is to be traversed. Specify one of the following: • `"firstSwitchList"`: As the design is traversed, DesignSync DFII descends into the first view specified in the switch list that exists for a cell. Specify the switch list using the `l_switchList` argument. (Default) • `"allSwitchList"`: As the design is traversed, DesignSync DFII descends into each view of the cell in the workspace that matches a view in the switch list. Specify the switch list using the `l_switchList` argument. • `"instantiatedView"`: As the design is traversed, DesignSync DFII descends into each instantiated view. The `l_switchList` argument is ignored in this case. • `"allViews"`: As the design is traversed, DesignSync DFII descends into each view of the cell

	in the workspace that exists for each cell. The `l_switchList` argument is ignored in this case.
`l_switchList`	Names of the views to be scanned to identify the design hierarchy. The `l_switchList` argument is required if you specify the `"firstSwitchList"` or `"allSwitchList"` values using the `t_switchUsing` argument. If the `t_switchUsing` argument is set to `"instantiatedView"` or `"allViews"`, this argument is ignored.
`l_stopList`	Names of views at which the hierarchy scanning should stop. As the design is traversed, if the `l_switchList` view being scanned is also in this list, scanning stops.
`S_switchLibChoice`	Specifies which libraries to enter as the hierarchy is scanned: • `all`: Enter all libraries. (Default) • `only`: Enter only the libraries specified by the l`_switchLibNames` argument. • `not`: Enter all libraries except those specified by the l`_switchLibNames` argument.
`l_switchLibNames`	Library names controlled by the `S_switchLibChoice` argument. You need not include this argument if `all` is selected as the `S_switchLibChoice` argument.
`g_processViews`	Once you have identified the hierarchy using the `t_switchUsing` argument, as well as the switch list and stop list if necessary, specify the views of the identified cells to be processed: • `t`: Process all views that exist for the cell. • `nil`: Process only the single view switched into. (Default) • List of views to process.
`gS_processFiles`	Specifies whether cell- and library-level files are processed in addition to the specified cell views: • `nil`: No cell- or library-level files are processed. (Default) • `cell`: Cell-level files are processed, but library-level files are not. This option selects only cell-level files for those cells on which you are operating. • `library`: Cell- and library-level files are processed.

`t_tag`	Tags the object version or module version on the server with the specified tag name. For module objects, all objects are evaluated before the checkin begins. If the module cannot be tagged, for example if the user does not have access to add a tag or because the tag exists and is immutable, the entire module checkin fails. **Note:** Individual module objects cannot have tags. Only the module itself can be tagged.
`t_mode`	Check-in mode (`lock` , `share`, `mirror`, or `keep`). By default, the mode matches the default fetch mode. See DesignSync DFII Help:Selecting a Default Fetch Mode to learn how to set the default fetch mode.
`g_silent`	Run silently (`t`). (Default)
`t_comment`	Check-in comment. By default, no check-in comment is supplied. However, if DesignSync DFII has been configured to require a comment of a particular length, a check-in comment is required.
`g_force`	Overwrite locally modified files in your workspace (`t`). By default, local changes are not overwritten when you check out files (`nil`).
`g_iflock`	Specifies whether to check in all modified objects in the workspace (f) (default) or only targeted files (t). Targeted files include: locked DesignSync vault files or module members and added, renamed, or removed module objects.
`g_new`	Allow new (or retired) items to be checked in (`t`). By default, checking in new items is not allowed (`nil`).
`g_skip`	Skip version (`t`). By default, version skipping is not allowed (`nil`).
`g_retain`	Retain the "last modified" timestamps of the objects that remain in your workspace (`t`), or make the timestamps the check-in time (`nil`). The default is `nil` unless defined otherwise from SyncAdmin (see SyncAdmin Help). The retain option is only meaningful when leaving physical copies of objects in your workspace (`lock` and `keep` modes), and is silently ignored otherwise.
`t_branch`	Checks the object into the specified branch, rather than checking the object into the branch it was checked out from. The branch tag can be any valid branch selector, including auto(*branchname*).

	Note: This option is not applicable to modules. If used with the -modulecontext option, or on module data, the command fails with an appropriate error.
`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.

Value Returned

Returns a list of pass and fail counts; the first integer represents the number of objects successfully checked in and the second integer represents the number of failures. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssCheckinLibraryP

```
dssCheckinLibraryP(
  t_libName [?viewNames l_viewNames] [?mode t_mode]
  [?force g_force][?comment t_comment] [?skip g_skip]
  [?new g_new] [?retain g_retain] [?silent g_silent]
  [?background g_background] [?moduleContext t_moduleContext]
  [?branch t_branch] [?iflock g_lock] [?tag g_tag]
)
=> nil/(x_pass x_fail)
```

Description

Checks in a library, either all the objects in the library or a specified list of cell views.

Arguments

`t_libName`	Library name. (Required)
`l_viewNames`	One or more view name(s) to be checked in. (Optional). Checks in all views by default.
`t_mode`	Check-in mode ("`lock`", "`share`", "`mirror`", or "`keep`"). By default, the mode matches the default fetch mode. See the DesignSync Data Manager DFII User's Guide:Selecting a

	Default Fetch Mode to learn how to set the default fetch mode.
`g_force`	Force a checkin to create new versions in the vault (`t`). The default is `nil`.
`t_comment`	Check-in comment. By default, no check-in comment is supplied. However, if DesignSync DFII has been configured to require a comment of a particular length, a check-in comment is required.
`g_skip`	Skip version (`t`). By default, version skipping is not allowed (`nil`).
`g_new`	Allow new (or retired) items to be checked in (`t`). By default, checking in new items is not allowed (`nil`).
`g_retain`	Retain the "last modified" timestamps of the objects that remain in your workspace (`t`), or make the timestamps the check-in time (`nil`). The default is `nil` unless defined otherwise from SyncAdmin (see SyncAdmin Help). The retain option is only meaningful when leaving physical copies of objects in your workspace (`lock` and `keep` modes), and is silently ignored otherwise.
`g_silent`	Run silently (`t`). (Default)
`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.
`t_moduleContext`	The module context to restrict the operation to. The module must have its base directory at, or above, the level of the library being checked in. **Note:** You can only specify one module. If you are checking in objects to two different modules in the same workspace, use two separate checkin operations. If you do not specify a module context, the operation applies to all objects specified.
`t_branch`	Checks the object into the specified branch, rather than

	checking the object into the branch it was checked out from. The branch tag can be any valid branch selector, including auto(*branchname*). **Note:** This option is not applicable to modules. If used with the -modulecontext option, or on module data, the command fails with an appropriate error.
`g_iflock`	Specifies whether to check in all modified objects in the workspace (f) (default) or only targeted files (t). Targeted files include: locked DesignSync vault files or module members and added, renamed, or removed module objects.
`t_tag`	Tags the object version or module version on the server with the specified tag name. For module objects, all objects are evaluated before the checkin begins. If the module cannot be tagged, for example if the user does not have access to add a tag or because the tag exists and is immutable, the entire module checkin fails. **Note:** Individual module objects cannot have tags. Only the module itself can be tagged.

Value Returned

Returns a list of pass and fail counts; the first integer represents the number of objects successfully checked in and the second integer represents the number of failures. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssCheckoutCategoryP

```
dssCheckoutCategoryP(
  t_libName tl_catNames [?viewNames l_viewNames]
  [?mode t_mode] [?tag t_tag] [?force g_force]
  [?overlay g_overlay][?nested g_nested] [?all g_all]
[?retain g_retain] [?unifyState g_unifyState]
[?silent g_silent] [?background g_background]
)
> nil/(x_pass x_fail)
```

Description

Checks out objects of one or more categories. You can check out all the objects in a category at one time or specify views to check out.

Arguments

`t_libName`	Library name. (Required)
`tl_catNames`	One or more category names. (Required)
`l_viewNames`	One or more view name(s) to be checked out. (Optional). Checks out all views by default.
`t_mode`	Check-out mode ("`lock`", "`share`", "`mirror`", "`get`", or "`lockref`"). By default, the mode matches the default fetch mode. See the DesignSync Data Manager DFII User's Guide:Selecting a Default Fetch Mode to learn how to set the default fetch mode.
`t_tag`	Selector, or version name, to be checked out. By default, no selector is specified, in which case the persistent selector list determines the version -- typically the latest version on the current branch. See DesignSync Data Manager User's Guide to learn more about selectors. **Note:** When used with modules, this identifies the module version, not the module member version. Use the `t_ version` option to specify a module member version. `t_version` and `t_tag` are mutually incompatible.
`g_force`	Overwrite locally modified files in your workspace (`t`). By default, local changes are not overwritten when you check out files (`nil`).
`g_overlay`	Fetch a version of a design object from another branch and overlay it on the version you have checked out in your workspace (`t`). By default, an overlay is not performed (`nil`). **Note**: This option is available only if `t_mode` is set to "`get`".
`g_nested`	Apply to nested category contents (`t`). (Default) **Note:** If `g_nested` is set to `t` but one or more nested category files are missing from your workspace, DesignSync DFII automatically fetches the missing category files and processes the specified objects.
`g_all`	Check out all matching category objects (`t`), even those objects that are not in your local workspace. (Default) If set to `nil`, checks out only those objects that are already in your local workspace.
`g_retain`	Retain the "last modified" timestamps of the checked-out objects as recorded when the object was checked into the vault (`t`), or make the timestamps the check-out time (`nil`). The default is `nil` unless defined otherwise from SyncAdmin (see SyncAdmin Help).

	The retain option is only meaningful when checking out physical copies (`lock` and `get` modes) and is silently ignored otherwise.
`g_unifyState`	Put objects in the state specified by `t_mode` even if the workspace already contains the requested version and therefore no checkout is required (`t`). By default (`nil`), a check-out operation only changes the states of objects that are fetched from the vault.
`g_silent`	Run silently (`t`). (Default)
`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.

Value Returned

Returns a list of pass and fail counts; the first integer represents the number of objects successfully checked out and the second integer represents the number of failures. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssCheckoutCellP

```
dssCheckoutCellP(
  t_libName tl_cellNames [?viewNames l_viewNames]
  [?mode t_mode] [?tag t_tag] [?force g_force]
[?overlay g_overlay] [?retain g_retain]
[?unifyState g_unifyState] [?silent g_silent]
[?background g_background]
)
=> nil/(x_pass x_fail)
```

Description

Checks out one or more cells, either all the objects in each cell or a specified set of cell views.

Arguments

`t_libName`	Library name. (Required)

`tl_cellNames`	One or more cell name(s) to be checked out. (Required)
`l_viewNames`	One or more view name(s) to be checked out. (Optional). Checks out all views by default.
`t_mode`	Check-out mode ("`lock`", "`share`", "`mirror`", "`get`", or "`lockref`"). By default, the mode matches the default fetch mode. See the DesignSync Data Manager DFII User's Guide:Selecting a Default Fetch Mode to learn how to set the default fetch mode.
`t_tag`	Selector, or version name, to be checked out. By default, no selector is specified, in which case the persistent selector list determines the version -- typically the latest version on the current branch. See DesignSync Data Manager User's Guide to learn more about selectors. **Note:** When used with modules, this identifies the module version, not the module member version.
`g_force`	Overwrite locally modified files in your workspace (`t`). By default, local changes are not overwritten when you check out files (`nil`).
`g_overlay`	Fetch a version of a design object from another branch and overlay it on the version you have checked out in your workspace (`t`). By default, an overlay is not performed (`nil`). **Note**: This option is available only if `t_mode` is set to "`get`".
`g_retain`	Retain the "last modified" timestamps of the checked-out objects as recorded when the object was checked into the vault (`t`), or make the timestamps the check-out time (`nil`). The default is `nil` unless defined otherwise from SyncAdmin (see SyncAdmin Help). The retain option is only meaningful when checking out physical copies (`lock` and `get` modes) and is silently ignored otherwise.
`g_unifyState`	Put objects in the state specified by `t_mode` even if the workspace already contains the requested version and therefore no checkout is required (`t`). By default (`nil`), a check-out operation only changes the states of objects that are fetched from the vault.
`g_silent`	Run silently (`t`). (Default)
`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and

Background Commands for information about the output of background commands.

Value Returned

Returns a list of pass and fail counts; the first integer represents the number of objects successfully checked out and the second integer represents the number of failures. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssCheckoutCellViewP

```
dssCheckoutCellViewP(
  t_libName t_cellName t_viewName
[?tag t_tag] [?vaultVersion t_vaultVersion]
  [?mode t_mode] [?force g_force] [?overlay g_overlay]
[?retain g_retain] [?unifyState g_unifyState]
[?silent g_silent] [?background g_background]
)
=> nil/(x_pass x_fail)
```

Description

Checks out a single cell view.

Arguments

`t_libName`	Library name. (Required)
`t_cellName`	Cell name. (Required)
`t_viewName`	View name. (Required)
`t_tag`	Selector, or version name, to be checked out. By default, no selector is specified, in which case the persistent selector list determines the version -- typically the latest version on the current branch. See DesignSync Data Manager User's Guide to learn more about selectors. **Note:** When used with modules, this identifies the module version, not the module member version. Use the `t_ version` option to specify a module member version. `t_version` and `t_tag` are mutually incompatible.
`t_vaultVersion`	Version number of the module member object to be checked out. **Note:** The `t_version` and `t_tag` options are mutually

	incompatible.
`t_mode`	Check-out mode ("`lock`", "`share`", "`mirror`", "`get`", or "`lockref`"). By default, the mode matches the default fetch mode. See the DesignSync Data Manager DFII User's Guide:Selecting a Default Fetch Mode to learn how to set the default fetch mode.
`g_force`	Overwrite locally modified files in your workspace (`t`). By default, local changes are not overwritten when you check out files (`nil`).
`g_overlay`	Fetch a version of a design object from another branch and overlay it on the version you have checked out in your workspace (`t`). By default, an overlay is not performed (`nil`). **Note**: This option is available only if `t_mode` is set to "`get`".
`g_retain`	Retain the "last modified" timestamp of the checked-out view view as recorded when the view was checked into the vault (`t`), or make the timestamp the check-out time (`nil`). The default is `nil` unless defined otherwise from SyncAdmin (see SyncAdmin Help). The retain option is only meaningful when checking out physical copies (`lock` and `get` modes) and is silently ignored otherwise.
`g_unifyState`	Put the view in the state specified by `t_mode` even if the workspace already contains the requested version and therefore no checkout is required (`t`). By default (`nil`), a check-out operation only changes the states of objects that are fetched from the vault.
`g_silent`	Run silently (`t`). (Default)
`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.

Value Returned

Returns a list of pass and fail counts; the first integer represents the number of objects successfully checked out and the second integer represents the number of failures. The `dssCheckoutCellViewP` function lets you check out a single cell view only, so the returned list is (1 0) if the checkout is successful and (0 1) if the checkout fails. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssCheckoutFileP

```
dssCheckoutFileP(
  tl_fileNames [?mode t_mode] [?tag t_tag]
  [?force g_force] [?incremental g_incremental]
[?overlay g_overlay] [?retain g_retain]
[?unifyState g_unifyState][?silent g_silent]
[?recursive g_recursive] [?background g_background]
[?moduleContext t_moduleContext]
)
=> nil/(x_pass x_fail)
```

Description

Checks out one or more file objects.

You can specify absolute or relative filenames to be checked out. Filenames can be relative to the current working directory or to any library on the library path. For example, if library `acc` is on your library path, then you can specify the `cdsinfo.tag` file for that library as `acc/cdsinfo.tag`, even though the `acc` library directory might be anywhere on disk. If a library name exists, and there is also a directory within the current working directory of the same name, the library name is used.

Specify wildcards for filenames using glob-style expressions.

Note:

For wildcards, filenames in the current working directory take precedence over library names. That is, a glob expression of `lib*` will not match libraries `libA` and `libB` if similarly named files exist in the current working directory; the `dssCheckoutFileP` function first expands regular expressions against the current directory, and then performs library matching.

Arguments

`tl_fileNames`	One or more file object(s) to be checked out. (Required) You can specify file objects as glob-style expressions. A file object can be: A filename, specified as a full path or a path relative to the current working directory.

	A filename, specified relative to a library, for example `<libname>/cdsinfo.tag` or `<libname>/cellname/prop.xx`. A directory name, either a full path or a path relative to the current working directory. A library name. A cell name, specified as `<libname>/<cellname>`. A view name, specified as `<libname>/<cellname>/<viewname>`. **Note**: DesignSync creates objects called <name>`.sync.cds` to represent Cadence views, where <name> corresponds to the name of the view folder, for example: `~/ttlLib/and2/symbol.sync.cds`. These objects are not actual files; thus, you cannot apply the `dssCheckoutFileP` function to this type of object.
`t_mode`	Check-out mode ("`lock`", "`share`", "`mirror`", "`get`", or "`lockref`"). By default, the mode matches the default fetch mode. See DesignSync DFII Help: Selecting a Default Fetch Mode to learn how to set the default fetch mode. **Note:** Because `lockref` mode should only be used when you are regenerating data, it is not an appropriate mode for all object types. For example, `lockref` is likely not appropriate for library-level files such as `prop.xx`, `cdsinfo.tag`, and category files.
`t_tag`	Selector, or version name, to be checked out. By default, no selector is specified, in which case the persistent selector list determines the version -- typically the latest version on the current branch. See DesignSync Data Manager User's Guide to learn more about selectors. **Note:** When used with modules, this identifies the module version, not the module member version.
`g_force`	Overwrite locally modified objects in your workspace (`t`). By default, local changes are not overwritten when you check out objects (`nil`).
`g_incremental`	Perform incremental (`t`) or full (`nil`) populate. If not provided, incremental behavior is determined by SyncAdmin setting (see SyncAdmin Help).

`g_overlay`	Fetch a version of an object from another branch and overlay it on the version you have checked out in your workspace (`t`). By default, an overlay is not performed (`nil`). **Note**: This option is available only if `t_mode` is set to "`get`".
`g_retain`	Retain the "last modified" timestamps of the checked-out objects as recorded when the object was checked into the vault (`t`), or make the timestamps the check-out time (`nil`). The default is `nil` unless defined otherwise from SyncAdmin (see SyncAdmin Help). The retain option is only meaningful when checking out physical copies (`lock` and `get` modes) and is silently ignored otherwise.
`g_unifyState`	Put objects in the state specified by `t_mode` even if the workspace already contains the requested version and therefore no checkout is required (`t`). By default (`nil`), a check-out operation only changes the states of objects that are fetched from the vault.
`g_silent`	Run silently (`t`). (Default)
`g_recursive`	Check out all objects in each specified directory, as well as its subdirectories (`t`). (Default)
`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.
`t_moduleContext`	The module context to restrict the operation to. The module must have its base directory at, or above, the level of the object being checked out. If you do not specify a module context, the operation applies to all objects specified. **Note:** You can only specify one module. If you are checking out objects to two different modules in the same workspace, use two separate checkout operations.

Value Returned

Returns a list of pass and fail counts; the first integer represents the number of objects successfully checked out and the second integer represents the number of failures. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssCheckoutHierarchyP

```
dssCheckoutHierarchyP(
  t_libName t_cellName tl_viewNames
  [?switchUsing t_switchUsing]
  [?switchList l_switchList] [?mode t_mode]
  [?force g_force] [?tag t_tag]
  [?overlay g_overlay] [?retain g_retain]
  [?unifyState g_unifyState] [?stopList l_stopList]
  [?fetchMissingCells g_fetchMissingCells]
  [?switchLibChoice S_switchLibChoice]
  [?switchLibNames l_switchLibNames]
  [?processViews gl_processViews]
  [?processFiles gS_processFiles] [?silent g_silent]
  [?background g_background]
)
=> nil/(x_pass x_fail)
```

Description

Checks out objects of a design hierarchy. To identify the cells in a design hierarchy, DesignSync DFII scans the hierarchy, beginning with the top-level cell views you specify using the `tl_viewNames` argument. Then, DesignSync DFII descends into the views indicated by the `t_switchUsing` argument. You can use the `t_switchUsing` argument to specify that DesignSync DFII descend into one or more views you specify in a switch list (using the `l_switchList` argument). You can instead have DesignSync DFII descend into all instantiated views or all views that exist for a cell by setting the `t_switchUsing` argument to `"instantiatedView"` or `"allViews"`, respectively. Use the `l_stopList` argument to indicate at which views DesignSync DFII is to stop scanning. DesignSync DFII also offers other hierarchy controls, such as limiting which libraries are scanned using the `S_switchLibChoice` argument and limiting which views are checked in using the `g_processViews` argument.

Notes:

- For DesignSync DFII to scan the hierarchy, the cells must be in your local workspace.

- DesignSync DFII does not scan through libraries that have been filtered out using the `l_switchLibNames` argument. For example, suppose a cell in **library_1** references a cell in **library_2**, which references a cell in **library_3**. If **library_2** is filtered out in the `l_switchLibNames` argument, the cell in **library_3** is not found.

Arguments

`t_libName`	Top library name of hierarchy to be checked out. (Required)
`t_cellName`	Top cell name of hierarchy to be checked out. (Required)
`tl_viewNames`	Top-level view names of hierarchies to be checked out. (Required) Can be given a single view, a string, or a list of views.
`t_switchUsing`	Indicates how the design hierarchy is to be traversed. Specify one of the following: • `"firstSwitchList"`: As the design is traversed, DesignSync DFII descends into the first view specified in the switch list that exists for a cell. Specify the switch list using the `l_switchList` argument. (Default) • `"allSwitchList"`: As the design is traversed, DesignSync DFII descends into each view of the cell in the workspace that matches a view in the switch list. Specify the switch list using the `l_switchList` argument. • `"instantiatedView"`: As the design is traversed, DesignSync DFII descends into each instantiated view. The `l_switchList` argument is ignored in this case. • `"allViews"`: As the design is traversed, DesignSync DFII descends into each view of the cell in the workspace that exists for each cell. The `l_switchList` argument is ignored in this case.
`l_switchList`	Names of the views to be scanned to identify the design hierarchy. The `l_switchList` argument is required if you specify the `"firstSwitchList"` or `"allSwitchList"` values using the `t_switchUsing` argument. If the `t_switchUsing` argument is set to `"instantiatedView"` or `"allViews"`, this argument is ignored.
`t_mode`	Check-out mode ("`lock`", "`share`", "`mirror`", "`get`", or

	"`lockref`"). By default, the mode matches the default fetch mode. See DesignSync DFII Help:Selecting a Default Fetch Mode to learn how to set the default fetch mode.
`g_force`	Overwrite locally modified files in your workspace (`t`). By default, local changes are not overwritten when you check out files (`nil`).
`t_tag`	Selector, or version name, to be checked out. By default, no selector is specified, in which case the persistent selector list determines the version -- typically the latest version on the current branch. See DesignSync Data Manager User's Guide to learn more about selectors. **Note:** When used with modules, this identifies the module version, not the module member version.
`g_overlay`	Fetch a version of a design object from another branch and overlay it on the version you have checked out in your workspace (`t`). By default, an overlay is not performed (`nil`). **Note**: This option is available only if `t_mode` is set to "`get`".
`g_retain`	Retain the "last modified" timestamps of the checked-out objects as recorded when the object was checked into the vault (`t`), or make the timestamps the check-out time (`nil`). The default is `nil` unless defined otherwise from SyncAdmin (see SyncAdmin Help). The retain option is only meaningful when checking out physical copies (`lock` and `get` modes) and is silently ignored otherwise.
`g_unifyState`	Put objects in the state specified by `t_mode` even if the workspace already contains the requested version and therefore no checkout is required (`t`). By default (`nil`), a check-out operation only changes the states of objects that are fetched from the vault.
`l_stopList`	Names of views at which the hierarchy scanning should stop. As the design is traversed, if the `l_switchList` view being scanned is also in this list, scanning stops.
`g_fetchMissingCells`	Fetch cells in the hierarchy that are not present in the workspace (`t`). By default, a hierarchical checkout only fetches cells in your workspace (`nil`). Enabling this option also fetches missing views corresponding to the cells in the workspace. Select this option to check out the entire hierarchy, even

	if some cells are not currently in your workspace. The checkout can be significantly slower, but it ensures that the entire hierarchy is checked out. The checkout is iterative -- if cells or views are missing, DesignSync DFII fetches those objects and then scans the hierarchy again in order to fetch objects referenced by the missing views and cells. This iterative scanning continues until all of the missing cells and views have been fetched.
`S_switchLibChoice`	Specifies which libraries to enter as the hierarchy is scanned: • `all`: Enter all libraries. (Default) • `only`: Enter only the libraries specified by the l`_switchLibNames` argument. • `not`: Enter all libraries except those specified by the l`_switchLibNames` argument.
`l_switchLibNames`	Library names controlled by the `S_switchLibChoice` argument. You need not include this argument if `all` is selected as the `S_switchLibChoice` argument.
`g_processViews`	Once you have identified the hierarchy using the `t_switchUsing` argument, as well as the switch list and stop list if necessary, specify the views of the identified cells to be processed: • `t`: Process all views that exist for the cell. • `nil`: Process only the single view switched into. (Default) • List of views to process.
`gS_processFiles`	Specifies whether cell- and library-level files are processed in addition to the specified cell views: • `nil`: No cell- or library-level files are processed. (Default) • `cell`: Cell-level files are processed, but library-level files are not. This option selects only cell-level files for those cells on which you are operating. • `library`: Cell- and library-level files are processed.
`g_silent`	Run silently (`t`). (Default)
`g_background`	Run command in the background (`t`). By default,

commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue.

See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.

Value Returned

Returns a list of pass and fail counts; the first integer represents the number of objects successfully checked out and the second integer represents the number of failures. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssCheckoutLibraryP

```
dssCheckoutLibraryP(
  t_libName [?viewNames l_viewNames] [?mode t_mode]
  [?tag t_tag] [?force g_force] [?incremental g_incremental]
[?overlay g_overlay] [?retain g_retain]
[?unifyState g_unifyState][?silent g_silent]
[?background g_background] [?moduleContext t_moduleContext]
)
=> nil/(x_pass x_fail)
```

Description

Checks out a library, either all the objects in the library or a specified list of cell views.

Note:

By default, a library checkout only fetches the views corresponding to the cells currently in the workspace. If you want to be sure to fetch the specified views for each cell in the library and not just those in the workspace, enable the `syncFetchMissingCellsOnLibViewsCheckout` SKILL variable. See the DesignSync Data Manager DFII User's Guide: Fetching Views of Missing Cells During Library Checkout for details.

Arguments

`t_libName`	Library name. (Required)
`l_viewNames`	One or more view name(s) to be checked out. (Optional).

	Checks out all views by default.
`t_mode`	Check-out mode ("`lock`", "`share`", "`mirror`", "`get`", or "`lockref`"). By default, the mode matches the default fetch mode. See the DesignSync Data Manager DFII User's Guide: Selecting a Default Fetch Mode to learn how to set the default fetch mode. **Note:** Because `lockref` mode should only be used when you are regenerating data, it is not an appropriate mode for all object types. For example, `lockref` is likely not appropriate for library-level files such as `prop.xx`, `cdsinfo.tag`, and category files.
`t_tag`	Selector, or version name, to be checked out. By default, no selector is specified, in which case the persistent selector list determines the version -- typically the latest version on the current branch. See DesignSync Data Manager User's Guide to learn more about selectors. **Note:** When used with modules, this identifies the module version, not the module member version.
`g_force`	Overwrite locally modified files in your workspace (`t`). By default, local changes are not overwritten when you check out files (`nil`).
`g_incremental`	Perform incremental (`t`) or full (`nil`) populate. If not provided, incremental behavior is determined by SyncAdmin setting (see SyncAdmin Help).
`g_overlay`	Fetch a version of a design object from another branch and overlay it on the version you have checked out in your workspace (`t`). By default, an overlay is not performed (`nil`). **Note**: This option is available only if `t_mode` is set to "`get`".
`g_retain`	Retain the "last modified" timestamps of the checked-out objects as recorded when the object was checked into the vault (`t`), or make the timestamps the check-out time (`nil`). The default is `nil` unless defined otherwise from SyncAdmin (see SyncAdmin Help). The retain option is only meaningful when checking out physical copies (`lock` and `get` modes) and is silently ignored otherwise.
`g_unifyState`	Put objects in the state specified by `t_mode` even if the workspace already contains the requested version and therefore no checkout is required (`t`). By default (`nil`), a check-out operation only changes the states of objects that are fetched from the vault.

`g_silent`	Run silently (`t`). (Default)
`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.
`t_moduleContext`	The module context to restrict the operation to. The module must have its base directory at, or above, the level of the library being checked out. If you do not specify a module context, the operation applies to all objects specified. **Note:** You can only specify one module. If you are checking out objects to two different modules in the same workspace, use two separate checkout operations.

Value Returned

Returns a list of pass and fail counts; the first integer represents the number of objects successfully checked out and the second integer represents the number of failures. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssCompareViewsP

```
dssCompareViewsP(
  d_cv1 d_cv2 [?fileName t_outFile] [?silent g_silent]) =>
x_diffs
```

Description

Given two cell view, reports the differences between the two views to the specified file and optionally to the screen, and returns a count of the number of differences found.

Arguments

`d_cv1`	First View – a database reference as returned, for example, by `ddGetObj`. (Required)
`d_cv2`	Second View – a database reference as returned, for example,

by `ddGetObj`. (Required)

`t_outFile`

Name of the output file in which to record the differences. If the file already exists, then it is appended to by this operation. This allows the results of multiple comparisons to be written to the same file.

The file contents use the SKILL DPL (disembodied property list.) format. The list begins with the value nil and has alternating property names and values. The file is readable in SKILL using the standard lineread functions. The differences are contained in a sublist that uses a "reporting key" notation to record the location and type of the differences and allow you to locate them.

The properties in the list are:

- libName1: Library for first view
- cellName1: Cell for first view
- viewName1: View for first view
- libName2: Library for second view
- cellName2: Cell for second view
- viewName2: View for second view
- date: Date/time file was written (aka: command ran time) in the standard Cadence getCurrentTime() format which does not include the timezone.
- diffs: List of differences for each logical object type and each LPP. The differences list appears in the following order:

1. logical type (string, e.g. “inst”) or LPP (list of two elements, name and purpose)
2. list of objects in first view only where each item is identified by the reporting key for the item.
3. list of objects in second view onlywhere each item is identified by the reporting key for the item.
4. list of objects in both views that have differences, where each item is identified by the reporting key for the item and a list of the text descriptions of the differences.

`g_silent`

Run silently (`t`). (Default)

Note: If a value of nil is given, then a textual report of the differences is written to the screen.

Value Returned

`x_diffs`	The number of differences reported.

Example

This example shows results of comparing two versions of the inv/layout view where there is one difference in an “Instance” where a single property has changed.

```
dssCompareViewsP("layout" "layout_v1.2" ?fileName "diff")
```

"1"

Below is the resulting output file for the command.

Note: the file is written using a standard Cadence list-writing function which performs a “pretty print” of the list which spreads it across multiple lines and performs standard indentation..

```
(nil libName1 “master” cellName1 “inv”
viewName1 “layout” libName2 “master” cellName2
“inv” viewName2 “layout_v1.2” diffs (
    (“Instances” nil nil
        (((“basic” “pmos” “layout” 1.0 2.0) (“Property w changed
from 4.0 to 5.0”)))
    )
)
)
```

Usage Tips

The ability to programmatically compare views and report differences can be used in a variety of useful ways. A few sample usage scenarios are included below.

Simple check for no changes

The API functions can be used to check whether two versions of a view are identical by using the `dssFetchCellViewVersionP()` function to fetch the second version and then calling `dssCompareViewsP()` and checking for a `0` return value.

Simplified check for instance additions and removals

The API function can be used to compare the views and only report additions/removals by calling the `dssCompareViewsP()` function, and then reading in the results file (using SKILL) and processing the list to only report instances that exist only in the first view or only in the second view.

Check for differences across all views in a library with older release

You can populate two different releases of a library to two different areas (by populating both to different areas, and setting up the cds.lib file to use different library names.) and write a routine that compares the views in the two libraries. For any view that is in both libraries call `dssCompareViewsP()` to get a full description of all changes made in the data between two releases.

Related Topics

dssCompareViewsHandlerP

dssGetViewVersionsP

dssCompareViewsHandlerP

```
dssCompareViewsHandlerP(t_objType ?name t_name ?keygenProc
s_keygen ?reportKeyProc s_reportKey ?compareProc s_compare
?diffProc ls_diff ?extractProc s_extract ?figsProc s_figsProc)

=> t
```

Description

The `dssCompareViewsHandler` allows users to extend and modify the system to their own requirements. It allows customization of the way logical objects (instances, nets etc) and physical shapes (lines, paths, labels etc) are found, compared and reported. The keyed arguments are all optional, and if any keyed argument is not given then the default is used.

The included `dssCompareViewsP` API performs a useful comparison of views by default, and the `dssCompareViewsHandlerP` extension capability is only intended for users with specialized needs.

Note: If registering a handler for a new logical type (for example, pins) you must register a keygenProc, compareProc and extractProc otherwise the comparison will fail.

Arguments

t_objType	String. For a shape, this is the Cadence defined objType of the shape, for example “rect," “path,” or “label." For logical objects, this may be any string, though it is recommended that it match the Cadence objType where possible. If it is a logical object, and you want to modify the default handlers it must be one of

	“inst," “net,” or “terminal.” (Required) **Note:** See Cadence documentation for the full set of shape types.
`t_name`	String. The results display string for the type. This is for logical items only. This is the name that appears in the drop-down Results field on the GUI interface. This is optional and if not specified for a new logical object will default to the objType value. (Required)
`s_keygen`	Symbol. The name of a procedure to generate the key for an object. The procedure must take a single argument, d_object, which is the object for which a key is required, and must return a value (any type) that is the key for that object. **Note:** The default key is the objects bounding box.
s_reportKey	Symbol. The name of a procedure to generate the reporting key for an object. The procedure takes three arguments: t_objType (the object type), d_object (the object itself), and g_key (the key for the object.) The procedure must return a single value (any type) that is the reporting key for the object. **Note:** When reporting an object that has changed, the object passed to this routine is the object from the first cellview. The default routine returns a list of the object type and the key passed in. DesignSync recommends that this routine returns a list that starts with the object type.
s_compare	Symbol. The name of a procedure to compare two objects. The procedure takes two arguments: d_obj1 (the object in the first view) and d_obj2 (the object in the second view.) The procedure must return nil if the objects are considered to have no differences, otherwise it must return a list of strings detailing the differences. The strings may be any length, but long strings may not display well in the GUI results. The strings should NOT contain newline characters. The default is a routine that compares the user properties for the objects.
ls_diff	Symbol or list. May be a list of two strings, that report objects that are only in the first view or only in the second view. Otherwise, may be a procedure that takes two arguments, d_object and g_isFirst (true if the object is in the first view only, nil if in the second view only), and must return a single string that is used to report the objects present in only one view. The default value is the string pair “Only in first view” and “Only in second view”.
s_extract	Symbol. The name of a procedure to extract all object of a type from the cellview. The procedure takes two arguments: d_cv

	(one of the two views being compared), t_objType (the type of objects to be returned.) The procedure returns a list of the objects of the given type to be compared from the given view. This procedure is required for non-Shape types.
s_figsProc	Symbol. The name of a procedure to extract the figure(s) associated with an object. The procedure takes a single argument, d_object, and returns a list of the figures associated with that object, or a single figure. This procedure is required for non-Shape types to identify the figures that will be hilighted when a difference in this object is identified.

Value Returned

Returns t if the handler has been created; otherwise, returns nil.

Example

This example compares the “instHeaders” of the view, rather than the instances themselves, to see if any new types of sub-objects are being used. (instHeaders encapsulate the set of instances of the same master cellview that are instantiated.)

If there are new types, all instances using the new type are hilited. This also reports the lib/cell/view of the master cellview, and the number of instances that exist. There is no compare routine used because if the instHeader is in both views it is considered identical.

The registration function call is:

```
dssCompareViewsHandlerP(“instHeader” ?name “Master View”
?keygenProc ‘myKeyGen ?reportKeyProc ‘myReportKey ?compareProc
‘myCompareProc ?diffProc ‘myDiffProc ?extractProc ‘myExtractProc
?figsProc ‘myFigsProc)
```

The various functions used are:

```
procedure(myKeyGen(obj) list(obj~>libName obj~>cellName
obj~>viewName))

procedure(myReportKey(objType object key) sprintf(nil “%s:%s:%s”
object~>libName object~>cellName object~>viewName))

procedure(myCompareProc(obj1 obj2) nil)

procedure(myDiffProc(object isFirst) sprintf(nil “%d instances
only in %s cellview” length(object~>instances) if(isFirst
“first” “second”)))
```

```
procedure(myExtractProc(cv objtype) cv~>instHeaders)

procedure(myFigsProc(object) object~>instances)
```

Notes:

- You can modify the default handlers used by the system or reuse the functions in those handlers in your own extensions. The `dssCompareViewsListHandlersP` function can be used to indentify the default handlers.
- The keygenProc identifies the master using lib, cell and view names.
- The reportKeyProc was not really needed as the procedure could use the key.
- The compareProc simple returns nil to indicate two instHeaders with the same key are always considered to have no differences. A more complicated example could compare the properties of the instHeaders.
- The diffProc reports the number of instances of the master that exist. Ideally, this might want to allow for “variants” of the instHeader, for things like symbolic vias.
- The figsProc returns the list of instances of the instHeader. If you return instances in the figures list to the compare program automatically hilights the figures associated with the instances.

Related Topics

dssCompareViewsRemoveHandlerP

dssCompareViewsListHandlersP

```
dssCompareViewsListHandlersP()
```

Description

Shows a list of all the defined system and custom list handlers.

Arguments

None.

Return Value

Returns the defined list handlers in name/value pairs.

Related Topics

dssCompareViewsHandlerP

dssCompareViewsRemoveHandlerP

dssCompareViewsRemoveHandlerP

```
dssCompareViewsRemoveHandlerP(t_objType) => t
```

Description

Removes a defined custom handler created with dssCompareViewsHandlerP.

Argument

t_objType	String. This is the defined objType of the custom handler being removed. (Required)

Value Returned

Returns `t` if the handler has been removed; otherwise, returns `nil`.

Example

This example removes a custom defined shape object called "rect,"

```
dssCompareViewsRemoveHandlerP("rect")

t
```

dssConfigureLibraryP

```
dssConfigureLibraryP(
  t_libName [?vaultPath t_vaultPath]
  [?localMirrorPath t_localMirrorPath]
  [?selector t_selector] [?silent g_silent]
)
=> t/nil
```

Description

Configures the library for use with DesignSync DFII by setting the vault (data repository) and, optionally, the local mirror directory for the library.

Arguments

`t_libName`	Library name. (Required) **Note:** You cannot specify a module-managed library.
`t_vaultPath`	Vault URL for the data repository: `sync://<host>:<port>/Projects/<path>/<libraryName>` (for SyncServer vaults) `file:///<clientvaultpath>` (for client vaults) By default, the existing vault setting is used, if a vault has been previously set.
`t_localMirrorPath`	Path of the local mirror directory, for example, `/home/karen/mirrors/ASIC`. By default, the existing local mirror directory is used, if previously set.
`t_selector`	Selector specifying the branch of the library. For non-branching environments, specify the `Trunk` selector. By default, the selector is left unchanged.
`g_silent`	Run silently (`t`). (Default)

Note: To unset the value of an argument, supply the empty string ("").

Value Returned

Returns `t` if the library has been configured successfully; otherwise, returns `nil`.

dssCreateCellViewP

```
dssCreateCellViewP(
  t_libName t_cellName t_viewName t_toolName
  [?force g_force] [?moduleContext t_moduleContext]
)
=> t/nil
```

Description

Creates a cell view in the workspace and reserves the name in the vault. You reserve the name of the cell view at the time you create it to prevent other users from creating the same view, which could lead to data merging problems.

Arguments

`t_libName`	Library name. (Required)

`t_cellName`	Cell name. (Required)
`t_viewName`	View name. (Required)
`t_toolName`	Design tool with which to create the cell view, for example "`Composer-Schematic`". (Required). For a list of tool names, Select **Synchronicity => Create => Cell View** from the CIW. The Tool Name drop-down list displays the valid strings you specify as the `t_ToolName` argument.
`g_force`	Force a new cell view to be created locally even if the cell view already exists in the vault (`t`). By default, DesignSync DFII does not force the cell view to be created (`nil`).
`t_moduleContext`	The module context for the view being created. The module must have its base directory at, or above, the level of the object being created. The new view is automatically added to the selected module. **Note:** You can only specify one module.

Value Returned

Returns `t` if the cell view has been created; otherwise, returns `nil`.

dssDeleteCategoryP

```
dssDeleteCategoryP(
  t_libName tl_catNames [?stopOnError g_stopOnError]
  [?force g_force] [?vault g_vault] [?retire g_retire]
  [?remove g_remove] [?nested g_nested] [?silent g_silent]
  [?keepvid g_keepvid] [?background g_background]
)
=> t/nil
```

Description

Deletes objects of one or more categories from the workspace. You can also choose to delete or retire the objects from the vault.

Arguments

`t_libName`	Library name. (Required)

`tl_catNames`	One or more category names. (Required)
`g_stopOnError`	Cancel the delete operation if a delete operation fails for one of the objects (`t`). By default, DesignSync DFII continues with delete operations even if one of the delete operations fails (`nil`). The command output details any errors that might have occurred during the delete operations. **Note:** This option cannot be used with remove operations.
`g_force`	Force the objects to be deleted even if they are tagged or locked (`t`). By default, you cannot delete an object that is tagged or locked (`nil`).
`g_vault`	Delete the objects from the vault (`t`). By default, the objects are deleted only from your workspace (`nil`). **Note**: The `g_vault, g_retire`, and `g_remove` arguments are all mutually exclusive. If `g_vault` is set to `t` and `g_stopOnError` is set to `nil`, DesignSync DFII continues to delete the workspace object even if the vault object delete fails (for example, in the case where an access control is set).
`g_retire`	Retire the objects from the vault (`t`). By default, the objects are deleted only from your workspace (`nil`). **Note**: The `g_vault, g_retire`, and `g_remove` arguments are all mutually exclusive. If `g_retire` is set to `t` and `g_stopOnError` is set to `nil`, you might expect that DesignSync DFII will continue to delete the object from your workspace even if the retire from vault operation fails (for example, in the case where an access control is set). However, in some cases, if the retire operation fails, the object might remain in the workspace. **Note:** This option is not applicable modules and module members. If it is used on module or module members objects, the command fails.
`g_remove`	Removes the selected objects from the module (t). By default, the objects are deleted only from your workspace (`nil`). **Note**: The `g_vault, g_retire`, and `g_remove` arguments are all mutually exclusive. **Note:** This option is only applicable for module member objects. If it is used on any other type of object, including a module, the command fails.

`g_nested`	Apply to nested category contents (`t`). (Default) **Note:** If `g_nested` is set to `t` but one or more nested category files are missing from your workspace, DesignSync DFII automatically fetches the missing category files and processes the specified objects.
`g_silent`	Run silently (`t`). (Default)
`g_keepvid`	Retain information about the version ID of the Latest version in the vault (`t`). (Default)
`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.

Value Returned

Returns `t` if the objects of the named categories have been successfully deleted; otherwise, returns `nil`.

dssDeleteCellP

```
dssDeleteCellP(
  t_libName tl_cellNames [?stopOnError g_stopOnError]
  [?force g_force] [?vault g_vault] [?remove g_remove]
  [?retire g_retire] [?remove g_remove] [?silent g_silent]
[?keepvid g_keepvid] [?background g_background]
)
=> t/nil
```

Description

Deletes a cell and all its views from the workspace. You can also choose to delete or retire the cell from the vault.

Arguments

`t_libName`	Library name. (Required)
`tl_cellNames`	One or more cell name(s) to be deleted. (Required)
`g_stopOnError`	Cancel the delete operation if a delete operation fails for one of

	the objects (`t`). If you specify multiple cells, DesignSync DFII processes the cells in the order you list them in the `tl_cellNames` argument. By default, DesignSync DFII continues with delete operations even if one of the delete operations fails (`nil`). The command output details any errors that might have occurred during the delete operations. **Note:** This option cannot be used with remove operations.
`g_force`	Force the object or objects to be deleted even if they are tagged or locked (`t`). By default, you cannot delete an object that is tagged or locked (`nil`).
`g_vault`	Delete the object or objects from the vault (`t`). By default, objects are deleted only from your workspace (`nil`). **Note**: The `g_vault` and the `g_retire` arguments are mutually exclusive. If `g_vault` is set to `t` and `g_stopOnError` is set to `nil`, DesignSync DFII continues to delete the workspace object even if the vault object delete fails (for example, in the case where an access control is set).
`g_retire`	Retire the object or objects from the vault (`t`). By default, objects are deleted only from your workspace (`nil`). **Note**: The `g_vault` and the `g_retire` arguments are mutually exclusive. If `g_retire` is set to `t` and `g_stopOnError` is set to `nil`, you might expect that DesignSync DFII will continue to delete the object from your workspace even if the retire from vault operation fails (for example, in the case where an access control is set). However, in some cases, if the retire operation fails, the object might remain in the workspace.
`g_remove`	Removes the selected objects from the module (t). By default, the objects are deleted only from your workspace (nil). Note: The g_vault, g_retire, and g_remove arguments are all mutually exclusive. Note: This option is only applicable for module member objects. If it is used on any other type of object, including a module, the command fails.
`g_silent`	Run silently (`t`). (Default)
`g_keepvid`	Retain information about the version ID of the Latest version in the vault (`t`). (Default)

`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.

Value Returned

Returns `t` if the cell has been successfully deleted; otherwise, returns `nil`.

dssDeleteCellViewP

```
dssDeleteCellViewP(
  t_libName t_cellName t_viewName
  [?stopOnError g_stopOnError] [?force g_force]
  [?vault g_vault] [?retire g_retire] [?remove g_remove]
[?silent g_silent] [?keepvid g_keepvid]
[?background g_background]
)
=> t/nil
```

Description

Deletes a cell view from the workspace. You can also choose to delete or retire the cell view from the vault.

Arguments

`t_libName`	Library name. (Required)
`t_cellName`	Cell name. (Required)
`t_viewName`	View name. (Required)
`g_stopOnError`	Cancel the delete operation if part of the delete operation fails. (`t`). By default, DesignSync DFII continues with the delete operation even if part of the delete operation fails (`nil`). For example, if the `?vault` option is set to `t` and DesignSync DFII fails to delete the vault version of a cellview, you might still want DesignSync DFII to continue and delete the workspace version

	of the cellview. Note that if the `?retire` option is set to `t` and DesignSync DFII fails to retire the object, there are cases where the object remains in the workspace. The command output details any errors that might have occurred during the delete operations. **Note:** This option cannot be used with remove operations.
`g_force`	Force the cell view to be deleted even if it is tagged or locked (`t`). By default, you cannot delete an object that is tagged or locked (`nil`).
`g_vault`	Delete the cell view from the vault (`t`). By default, the cell view is deleted only from your workspace (`nil`). **Note**: The `g_vault` and the `g_retire` arguments are mutually exclusive. If `g_vault` is set to `t` and `g_stopOnError` is set to `nil`, DesignSync DFII continues to delete the workspace object even if the vault object delete fails (for example, in the case where an access control is set).
`g_retire`	Retire the cell view from the vault (`t`). By default, the cell view is deleted only from your workspace (`nil`). **Note**: The `g_vault` and the `g_retire` arguments are mutually exclusive. If `g_retire` is set to `t` and `g_stopOnError` is set to `nil`, you might expect that DesignSync DFII will continue to delete the object from your workspace even if the retire from vault operation fails (for example, in the case where an access control is set). However, in some cases, if the retire operation fails, the object might remain in the workspace.
`g_remove`	Removes the selected objects from the module (t). By default, the objects are deleted only from your workspace (nil). Note: The g_vault, g_retire, and g_remove arguments are all mutually exclusive. Note: This option is only applicable for module member objects. If it is used on any other type of object, including a module, the command fails.
`g_silent`	Run silently (`t`). (Default)
`g_keepvid`	Retain information about the version ID of the Latest version in the vault (`t`). (Default)

`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.

Value Returned

Returns `t` if the cell view has been successfully deleted; otherwise, returns `nil`.

dssDeleteFileP

```
dssDeleteFileP(
  tl_objectNames [?stopOnError g_stopOnError]
  [?force g_force] [?vault g_vault]
  [?retire g_retire] [?remove g_remove]
[?remCdsLib g_remCdsLib] [?silent g_silent]
[?keepvid g_keepvid] [?background g_background]
)
=> t/nil
```

Description

Deletes one or more file objects or directories.

Note: You cannot use this function to delete a module.

You can specify absolute or relative filenames or directory names to be deleted. Filenames and directory names can be relative to the current working directory or to any library on the library path. For example, if library `acc` is on your library path, then you can specify the `cdsinfo.tag` file for that library as `acc/cdsinfo.tag`, even though the `acc` library directory might be anywhere on disk. If a library name exists, and there is also a directory within the current working directory of the same name, the library name is used.

Specify wildcards for filenames and directory names using glob-style expressions.

Note:

For wildcards, filenames and directory names in the current working directory take precedence over library names. That is, a glob expression of `lib*` will not match libraries `libA` and `libB` if similarly named files or directories exist in the current working directory; the `dssDeleteFileP` function first expands regular expressions against the current directory, and then performs library matching.

Arguments

`tl_objectNames`	One or more file object(s) to be deleted. (Required) You can specify objects as glob-style expressions. An object can be: A filename, specified as a full path or a path relative to the current working directory. A directory, specified as a full path or a path relative to the current working directory. A filename, specified relative to a library, for example `<libname>/cdsinfo.tag` or `<libname>/cellname/prop.xx`. A library name. A cell name, specified as `<libname>/<cellname>`. A view name, specified as `<libname>/<cellname>/<viewname>`. **Note**: You cannot specify the type of view object that DesignSync creates, for example: `~/ttlLib/and2/symbol.sync.cds` as the filename. These objects are not actual files; thus, you cannot apply the `dssDeleteFileP` function to this type of object.
`g_stopOnError`	Cancel the delete operation if a delete operation fails for one of the objects (`t`). By default, DesignSync DFII continues with delete operations even if one of the delete operations fails (`nil`). The command output details any errors that might have occurred during the delete operations. **Note:** This option cannot be used with remove operations.
`g_force`	Force the object or objects to be deleted even if they are tagged or locked (`t`). By default, you cannot delete an object that is tagged or locked (`nil`).
`g_vault`	Delete the object or objects from the vault (`t`). By default,

	objects are deleted only from your workspace (`nil`). **Note**: The `g_vault` and the `g_retire` arguments are mutually exclusive. If `g_vault` is set to `t` and `g_stopOnError` is set to `nil`, DesignSync DFII continues to delete the workspace object even if the vault object delete fails (for example, in the case where an access control is set).
`g_retire`	Retire the object or objects from the vault (`t`). By default, objects are deleted only from your workspace (`nil`). **Note**: The `g_vault` and the `g_retire` arguments are mutually exclusive. If `g_retire` is set to `t` and `g_stopOnError` is set to `nil`, you might expect that DesignSync DFII will continue to delete the object from your workspace even if the retire from vault operation fails (for example, in the case where an access control is set). However, in some cases, if the retire operation fails, the object might remain in the workspace.
`g_remove`	Removes the selected objects from the module (t). By default, the objects are deleted only from your workspace (nil). Note: The g_vault, g_retire, and g_remove arguments are all mutually exclusive. Note: This option is only applicable for module member objects. If it is used on any other type of object, including a module, the command fails.
`g_remCdsLib`	Remove the library's entry from the `cds.lib` file (`t`). (Default)
`g_silent`	Run silently (`t`). (Default)
`g_keepvid`	Retain information about the version ID of the Latest version in the vault (`t`). (Default)
`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.

Value Returned

Returns `t` if the cell has been successfully deleted; otherwise, returns `nil`.

dssDeleteLibraryP

```
dssDeleteLibraryP(
  t_libName [?stopOnError g_stopOnError]
  [?force g_force] [?vault g_vault]
  [?retire g_retire] [?remove g_remove]
[?remCdsLib g_remCdsLib] [?silent g_silent]
[?keepvid g_keepvid] [?background g_background]
)
=> t/nil
```

Description

Deletes a library from the workspace. You can also choose to delete or retire the library from the vault.

Arguments

`t_libName`	Library name. (Required)
`g_stopOnError`	Cancel the delete operation if a delete operation fails for one of the objects (`t`). By default, DesignSync DFII continues with delete operations even if one of the delete operations fails (`nil`). The command output details any errors that might have occurred during the delete operations. **Note:** This option cannot be used with remove operations.
`g_force`	Force objects in the library to be deleted even if they are tagged or locked (`t`). By default, you cannot delete an object that is tagged or locked (`nil`).
`g_vault`	Delete the library from the vault (`t`). By default, the library is deleted only from your workspace (`nil`). **Note**: The `g_vault` and the `g_retire` arguments are mutually exclusive. If `g_vault` is set to `t` and `g_stopOnError` is set to `nil`, DesignSync DFII continues to delete the workspace object even if the vault object delete fails (for example, in the case where an access control is set).
`g_retire`	Retire the library from the vault (`t`). By default, the library is deleted only from your workspace (`nil`). **Note**: The `g_vault` and the `g_retire` arguments are mutually exclusive. If `g_retire` is set to `t` and `g_stopOnError` is set to `nil`, you might expect that DesignSync DFII will continue to delete the object from your workspace even if the retire from vault operation fails (for example, in the case where an access

	control is set). However, in some cases, if the retire operation fails, the object might remain in the workspace.
`g_remove`	Removes the selected objects from the module (t). By default, the objects are deleted only from your workspace (nil). Note: The g_vault, g_retire, and g_remove arguments are all mutually exclusive. Note: This option is only applicable for module member objects. If it is used on any other type of object, including a module, the command fails.
`g_remCdsLib`	Remove the library's entry from the `cds.lib` file (`t`). (Default)
`g_silent`	Run silently (`t`). (Default)
`g_keepvid`	Retain information about the version ID of the Latest version in the vault (`t`). (Default)
`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.

Value Returned

Returns `t` if the library has been successfully deleted; otherwise, returns `nil`.

dssDeleteTemporaryViewsP

```
dssDeleteTemporaryViewsP(
  t_libName [?silent g_silent]
)
=> t/nil
```

Description

Deletes any temporary cell views associated with the specified library. Temporary cell views are created when you fetch a cell view version using the `dssFetchCellViewVersionP` function.

Arguments

`t_libName`	Library name. (Required)

`g_silent`	Run silently (`t`). (Default)

Value Returned

Returns `t` if the temporary cell views are deleted successfully; otherwise, returns `nil`.

dssDeleteVersionP

```
dssDeleteVersionP(
  t_libName t_cellName t_viewName tl_versionNames
  [?force g_force] [?silent g_silent]
)
=> t/nil
```

Description

Deletes one or more versions of an object from the vault.

Note: Module versions cannot be deleted.

Arguments

`t_libName`	Library name. (Required)
`t_cellName`	Cell name. (Required)
`t_viewName`	View name. (Required)
`tl_versionNames`	One or more versions to be deleted. (Required)
`g_force`	Force versions to be deleted even if they are tagged or locked (`t`). By default, you cannot delete an object that is tagged or locked (`nil`).
`g_silent`	Run silently (`t`). (Default)

Value Returned

Returns `t` if the versions have been successfully deleted from the vault; otherwise, returns `nil`.

dssFetchCellViewVersionP

```
dssFetchCellViewVersionP(
  t_libName t_cellName t_viewName t_versionName
  [?open g_open] [?silent g_silent] [?moduleVersion
g_moduleversion]
)
=> t_versionName/nil
```

Description

Fetches a version of an object from the vault and creates a temporary view of the object, without affecting the workspace version of the object. You can also choose to open the temporary view of the version as part of the fetch operation. The function fetches the version only if it is not already available in a temporary view.

This function lets you compare two or more versions of the same cell view. DesignSync DFII creates a temporary, unmanaged copy of the specified version in your workspace. The name of the temporary cell view version is `<view>_v<version>`, where `<view>` is the name of the cell view, and `<version>` is the version number. For example, opening version 1.3 of cell view `layout` creates a temporary cell view called `layout_v1.3`.

To remove temporary views, use the `dssDeleteTemporaryViewsP` function.

Arguments

`t_libName`	Library name. (Required)
`t_cellName`	Cell name. (Required)
`t_viewName`	View name. (Required)
`t_versionName`	Version to be fetched. (Required)
`g_open`	Open the cell view version as part of the fetch operation (`t`). By default, the version is not opened automatically (`nil`).
`g_silent`	Run silently (`t`). (Default)
`?moduleVersion`	t/nil boolean value indicating whether the command uses the version as the module member vault version or the module version. The default, nil, indicates that the version specified is the module member vault version. This option is ignored for non-module objects.

Value Returned

If the version is fetched successfully, `dssFetchCellViewVersionP` returns the version number passed in or, if a selector is passed in, the version that corresponds to the selector; otherwise, returns `nil`.

dssFetchLockedP

```
dssFetchLockedP(
  t_objName [?vault g_vault] [?silent g_silent]
[?moduleContext t_moduleContext]
)
=>((l_object t_owner [t_branch t_time]) ...)/nil
```

Description

Reports the objects that are locked in the specified library or directory, returning a list of locked objects and their owners. You can specify whether to check for a lock on the objects in the local workspace or in the vault. For vault objects, the list includes the object's branch and the time the object was locked.

Arguments

`t_objName`	Library name, workspace module, or directory path. (Required)
`g_vault`	Check whether the vault versions of the objects are locked (`t`), and, for module members, what branch of the vault object is locked. By default, the `dssFetchLockedP` function checks whether the local workspace versions of the objects are locked (`nil`).
`g_silent`	Run silently (`t`). (Default)
`t_moduleContext`	The module context to restrict the operation to. The module must have its base directories at, or above, the level of the library being queried. If you do not specify a module context, the operation applies to all objects specified. **Note:** You can only specify one module with the modulecontext option.

Value Returned

Returns a list of locked objects and information about their locks. Each object's lock information is stored in a list that includes the object identifier sublist (`l_object`), the lock owner (`t_owner`) and, if the object is a vault object, the object's branch (`t_branch`) and lock time (`t_time`):

`l_object`	If the object is a library, `l_object` returns a list of the form (`t_library t_cell_name t_cell_view t_file`) The filename can be `nil`, indicating that the object is a cell view. The cell view can be `nil`, indicating a file at the cell level. Both the cell name and cell view can be `nil`, indicating a file at the

	library level. If the object is a directory, `l_object` returns a list of the form `(nil nil nil t_file)`indicating a file rather than a library object, where `t_file` is a filename relative to the specified directory (`l_object`). Note that if the directory contains libraries, `dssFetchLockedP` generates an entry for each locked library object, as well.
`t_owner`	The owner of the object.
`t_branch`	The name of the object's branch selector.
`t_time`	The date and time that the object was locked.

Returns `nil` if none of the objects in the library or directory are locked or if the library or directory is not under revision control. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

Example

The following examples show the return format of the `dssFetchLockedP` function. In `mylib`, Ian has the `cdsinfo.tag` file and the `mid2/schematic` view locked in his workspace. Fred has the `mid2/symbol` view locked. All locks are on branch `1` (Trunk):

```
dssFetchLockedP("mylib")
=>
(
  (("mylib" nil nil "cdsinfo.tag") "ian")
  (("mylib" "mid2" "schematic" nil) "ian")
)

dssFetchLockedP("mylib" ?vault t)
=>
(
(("mylib" nil nil "cdsinfo.tag") "ian" "1" "Fri Jul 06 16:42:51
BST 2001")
  (("mylib" "mid2" "schematic" nil) "ian" "1" "Mon Jul 09
08:45:28 BST 2001")
  (("mylib" "mid2" "symbol" nil) "fred" "1" "Thu Jul 05 13:21:05
BST 2001")
)
```

The following example shows the return format when a directory rather than a library is specified. The directory contains the `df2test` library; thus, the `dssFetchLockedP` function shows the locked objects in the `df2test` library, as well.

```
dssFetchLockedP("~/work")
=>
(((nil nil nil "readme.txt") "karen")
  (("df2test" nil nil "readme.txt") "karen")
  (("df2test" nil nil "cdsinfo.tag") "karen")
  (("df2test" "mid1" "verilog" nil) "karen")
  (("df2test" "mid1" "symbol" nil) "karen")
  (("df2test" "mid1" "schematic" nil) "karen")
  (("df2test" "mid1" "layout" nil) "karen")
)
```

The following example shows the return format when a directory is specified and the vault versions are queried.

```
dssFetchLockedP("~/work" ?vault t)
=>
(((nil nil nil "readme.txt") "karen" "1" "Thu Mar 28 13:02:30
EST 2002")
)
```

dssGetFileTagsP

```
dssGetFileTagsP(
  t_fileName
)
=> nil/l_tags
```

Description

Given a filename or workspace module name, returns a list of tags applied to the workspace version of the object.

Module objects can be specified by full path. For other DesignSync objects, you can specify an absolute or relative filename. Filenames can be relative to the current working directory or to any library on the library path. For example, if library `acc` is on your library path, then you can specify the `cdsinfo.tag` file for that library as `acc/cdsinfo.tag`, even though the `acc` library directory might be anywhere on disk. If a library name exists, and there is also a directory within the current working directory of the same name, the library name is used.

Arguments

`t_fileName`	A filename or workspace module name. (Required) A filename can be absolute or relative to the current working directory or to any library on the library path. **Note**: You must specify a

	filename; other file objects that resolve to directories, libraries, cells, and views are not supported by the `dssGetFileTagsP` function. Likewise, you cannot specify the type of view object that DesignSync creates, for example: `~/ttlLib/and2/symbol.sync.cds`. These objects are not actual files; thus, you cannot apply the `dssGetFileTagsP` function to this type of object.

Value Returned

`l_tags`	List of tags in reverse chronological order, with `Latest`, if present, always first in the list.

The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssGetFileVersionP

```
dssGetFileVersionP(
  t_fileName [?useCache g_useCache] [?quick g_quick]
)
=> t_version
```

Description

Given a filename or workspace module name, returns the version of the object in the workspace. By default, the `dssGetFileVersionP` function invokes the DesignSync `url versionid` command to determine the version. To improve performance, you can choose to access the cached or local metadata value of the object's version.

Module objects can be specified by full path. For other DesignSync objects, you can specify an absolute or relative filename. Filenames can be relative to the current working directory or to any library on the library path. For example, if library `acc` is on your library path, then you can specify the `cdsinfo.tag` file for that library as `acc/cdsinfo.tag`, even though the `acc` library directory might be anywhere on disk. If a library name exists, and there is also a directory within the current working directory of the same name, the library name is used.

Arguments

`t_fileName`	A filename or workspace module name. (Required) A filename can be absolute or relative to the current working directory or to any library on the library path. **Note**: You must specify a

	filename; other file objects that resolve to directories, libraries, cells, and views are not supported by the `dssGetFileVersionP` function. Likewise, you cannot specify the type of view object that DesignSync creates, for example: `~/ttlLib/and2/symbol.sync.cds`. These objects are not actual files; thus, you cannot apply the `dssGetFileVersionP` function to this type of object.
`g_useCache`	Return the existing cached value if one exists for the object's version (`t`). Extracting the cached value is the fastest method of obtaining the version. By default (`nil`), the `dssGetViewVersionP` function does not search for the cached value. If `g_useCache` is '`nil`' or the cached value is not found, the following other methods for extracting the version are attempted in this order: • If `g_quick` is '`t`', the local metadata value is returned. • If both `g_useCache` and `g_quick` are '`nil`', the DesignSync `url versionid` command is invoked. Note that the cached value of the version is automatically updated following any design management operation from the Synchronicity menu or any Auto-Checkin or Auto-Checkout operation.
`g_quick`	Return the local metadata value for the object's version (`t`). Quick mode also returns a fetch state indicator; use quick mode if you need to determine the fetch state as well as the version number. For locked objects, quick mode cannot provide the upcoming version; use the default to determine the upcoming version of a locked object. **Note**: If `g_useCache` and `g_quick` are both '`t`', the cached value is returned rather than the local metadata value. By default (`nil`), the `dssGetViewVersionP` function does not search for the local metadata value. If `g_quick` is '`nil`' or the local metadata value is not found, the following other methods for extracting the version are attempted in this order: • If `g_useCache` is '`t`', the cached value is returned. • If both `g_useCache` and `g_quick` are '`nil`', the DesignSync `url versionid` command is invoked.

Value Returned

`t_version`	The value returned depends upon the arguments selected: **Cached value used (useCache mode)**: Returns the cached value for the object's version if one exists, for example, `1.3`. If no cached value is found, quick mode is attempted next, and if the version cannot be obtained in quick mode, the DesignSync `url versionid` command is invoked. **Local metadata value used (quick mode)**: Returns the version number of the object followed by a fetch state indicator, for example, `1.3 (S)`. Fetch state indicators include: C: Cache mode M: Mirror mode L: Lock mode No state indicator: Fetch mode **Note**: In quick mode if a view is locked, only the current version is reported and not the upcoming versions, for example '`1.2 (L)`' is returned rather than '`1.2 -> 1.3`'. Also in quick mode, if the view is in mirror mode, then the value returned is always `‘Latest (M)’`, rather than a specific version number. **url versionid command invoked**: Returns the version number of the object, for example, `1.3`. If the object is locked, the current version number and upcoming version number are returned (`1.3 -> 1.4`)

dssGetFileVersionsP

```
dssGetFileVersionsP(
  t_fileName [?branchName t_branchName]
)
=> l_versions
```

Description

Given a filename or workspace module name, returns the list of versions that exist for that object, either all versions or those versions on a specified branch.

Module objects can be specified by full path. For other DesignSync objects, you can specify an absolute or relative filename. Filenames can be relative to the current working directory or to any library on the library path. For example, if library `acc` is on your library path, then you can specify the `cdsinfo.tag` file for that library as `acc/cdsinfo.tag`, even though the `acc` library directory might be anywhere on disk. If a library name exists, and there is also a directory within the current working directory of the same name, the library name is used.

Arguments

`t_fileName`	A filename or workspace module name. (Required) A filename can be absolute or relative to the current working directory or to any library on the library path. **Note**: You must specify a filename; other file objects that resolve to directories, libraries, cells, and views are not supported by the `dssGetFileVersionsP` function. Likewise, you cannot specify the type of view object that DesignSync creates, for example: `~/ttlLib/and2/symbol.sync.cds`. These objects are not actual files; thus, you cannot apply the `dssGetFileVersionsP` function to this type of object.
`t_branchName`	Branch name. Specify a branch selector, not a branch dot-numeric version number. You can also specify one of the following values for the `t_branchName` argument: • `all`: Returns all versions. (Default) • `current`: Returns all versions on the branch of the file currently in the workspace.

Value Returned

By default, the result is a list of all versions that exist in the vault for the specified object. If a branch name is specified using the `t_branchName` argument, the return list is restricted to the objects on that branch.

dssGetTagListP

```
dssGetTagListP(
  t_objName [?useCache g_useCache]
)
=> ((l_tags)/nil ((t_config t_tag) ...))/nil)
```

Description

Given a library, workspace module, or directory name, returns the user-defined tags and configuration tags for the object. User-defined tags can be defined in two places: the `syncUserTagList` SKILL variable and the DesignSync DFII Options form. Configuration tags are defined using ProjectSync. See the DesignSync Data Manager DFII User's Guide:Creating a Tag List for details.

Arguments

`t_objName`	Library, workspace module, or directory name. (Required)
`g_useCache`	Use existing cached configuration information to generate the configuration tags list (`t`, default). Using cached values is faster but may not reflect up-to-date information. If `g_useCache` is `nil` or cached information is not found, the DesignSync `url configs` command is invoked, which contacts the SyncServer for the latest configuration information. Note that `g_useCache` has no effect on the user-defined tags list; an up-to-date list is always returned.

Value Returned

Returns a two-element list. The first element is the list of user-defined tags, or `nil` if there are no user-defined tags. The second element is a list of two-element lists, each consisting of a configuration name and the corresponding vault tag, or `nil` if there are no configurations.

`l_tags`	Returns the list of user-defined tags from the registry and `syncUserTagList` variable.
`t_config`	Returns a configuration name.
`t_tag`	Returns the vault tag associated with the configuration name.

The function raises an error if argument checking fails.

Example

The following examples show the return format of the `dssGetTagListP` function.

In this example, the library `alib` has no user-defined tags or configurations:

```
dssGetTagListP("alib")
=> (nil nil)
```

In this example, the library `blib` has three user-defined tags (gold, silver, bronze) and two configurations (Alpha, Beta). The cached configuration information is not accessed in this example.

```
dssGetTagListP("blib" ?useCache nil)
=>
(("gold" "silver" "bronze")
 (("Alpha" "alpha_tag")
  ("Beta" "Beta")
 )
)
```

dssGetViewPathP

```
dssGetViewPathP(
  t_libName t_cellName t_viewName
)
=> nil/t_path
```

Description

Given a cell view, returns the DesignSync workspace path for that view object.

Arguments

`t_libName`	Library name. (Required)
`t_cellName`	Cell name. (Required)
`t_viewName`	View name. (Required)

Value Returned

`t_path`	Returns the path to the `view.sync.cds` object.

The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssGetViewTagsP

```
dssGetViewTagsP(
  t_libName t_cellName t_viewName
)
=> nil/l_tags
```

Description

Given a cell view, returns a list of tags applied to the workspace version of the view object.

Arguments

`t_libName`	Library name. (Required)
`t_cellName`	Cell name. (Required)
`t_viewName`	View name. (Required)

Value Returned

`l_tags`	List of tags in reverse chronological order, with `Latest`, if present, always first in the list.

The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssGetViewVersionP

```
dssGetViewVersionP(
  t_libName t_cellName t_viewName [?useCache g_useCache]
  [?quick g_quick] [?silent g_silent]
)
=> t_version
```

Description

Given a cell view, returns the version of the view object in the workspace. By default, the `dssGetViewVersionP` function invokes the DesignSync `url versionid` command to determine the version. To improve performance, you can choose to access the cached or local metadata value of the object's version.

Arguments

`t_libName`	Library name. (Required)
`t_cellName`	Cell name. (Required)
`t_viewName`	View name. (Required)
`g_useCache`	Return the existing cached value if one exists for the object's version (`t`). Extracting the cached value is the fastest method of obtaining the version. By default (`nil`), the `dssGetViewVersionP` function does not search for the cached value. If `g_useCache` is '`nil`' or the cached value is not found, the following other methods for extracting the version

are attempted in this order:

- If g_quick is 't', the local metadata value is returned.
- If both g_useCache and g_quick are 'nil', the DesignSync url versionid command is invoked.

Note that the cached value of the version is automatically updated following any design management operation from the Synchronicity menu or any Auto-Checkin or Auto-Checkout operation.

g_quick	Return the local metadata value for the object's version (t). Quick mode also returns a fetch state indicator; use quick mode if you need to determine the fetch state as well as the version number. For locked objects, quick mode cannot provide the upcoming version; use the default to determine the upcoming version of a locked object. **Note**: If g_useCache and g_quick are both 't', the cached value is returned rather than the local metadata value.

By default (nil), the dssGetViewVersionP function does not search for the local metadata value. If g_quick is 'nil' or the local metadata value is not found, the following other methods for extracting the version are attempted in this order:

- If g_useCache is 't', the cached value is returned.
- If both g_useCache and g_quick are 'nil', the DesignSync url versionid command is invoked.

g_silent	Run silently (t). (Default)

Value Returned

t_version — The value returned depends upon the arguments selected:

Cached value used (useCache mode): Returns the cached value for the object's version if one exists, for example, 1.3. If no cached value is found, quick mode is attempted next, and if the version cannot be obtained in quick mode, the DesignSync url versionid command is invoked.

Local metadata value used (quick mode): Returns the version number of the object followed by a fetch state indicator, for example, 1.3 (S). Fetch state indicators include:

C: Cache mode

M: Mirror mode

L: Lock mode

No state indicator: Fetch mode

Note: In quick mode if a view is locked, only the current version is reported and not the upcoming versions, for example '`1.2 (L)`' is returned rather than '`1.2 -> 1.3`'.

url versionid command invoked: Returns the version number of the object, for example, `1.3`. If the object is locked, the current version number and upcoming version number are returned (`1.3 -> 1.4`)

Example

The following example returns the version number of the `top` schematic:

```
dssGetViewVersionP("df2test" "top" "schematic")
=>
"1.1"
```

You can use the `dssGetViewVersionP` function to annotate a symbol so that its label contains the current version number of the associated schematic. To do so, you can create a label on the symbol of type, `iLLabel`, with `Label` value of:

```
dssGetViewVersionP(ilInst~>master~>libName
ilInst~>master~>cellName "schematic" ?useCache t ?quick t
?silent t) || "Unknown"
```

This call to `dssGetViewVersionP` passes in the library and cell names of the master of the symbol instance, as well as the `schematic` view name. The call requests the cache value for the version number and the 'quick' method. The quick method is important, as the `ILLabel` expression is evaluated each time the screen is refreshed, so it needs to be as fast as possible. If there is no associated `schematic` view, the `Label` value is set to `"Unknown"`; use of the `?silent` option ensures that only `"Unknown"` is returned as the `Label` value in this case.

dssGetViewVersionsP

```
dssGetViewVersionsP(
  t_libName t_cellName t_viewName
  [?branchName t_branchName]
)
=> l_versions
```

Description

Given a cell view, returns the list of versions that exist for that cell view, either all versions or those versions on a specified branch.

Arguments

`t_libName`	Library name. (Required)
`t_cellName`	Cell name. (Required)
`t_viewName`	View name. (Required)
`t_branchName`	Branch name. Specify a branch selector, not a branch dot-numeric version number. You can also specify one of the following values for the `t_branchName` argument: • `all`: Returns all versions. (Default) • `current`: Returns all versions on the branch of the version that is currently in the workspace.

Value Returned

By default, the result is a list of all versions that exist in the vault for the specified view. If a branch name is specified using the `t_branchName` argument, the return list is restricted to the versions on that branch.

dssIsFileLockedP

```
dssIsFileLockedP(
  t_fileName [?vault g_vault]
)
=> t_user/nil
```

Description

Reports whether the specified file object is locked, returning the lock owner if the file object is locked or `nil` if it is not locked. You can specify whether to check for a lock on the object in the local workspace or in the vault.

You can specify an absolute or relative filename. Filenames can be relative to the current working directory or to any library on the library path. For example, if library `acc` is on your library path, then you can specify the `cdsinfo.tag` file for that library as `acc/cdsinfo.tag`, even though the `acc` library directory might be anywhere on disk. If a library name exists, and there is also a directory within the current working directory of the same name, the library name is used.

Arguments

`t_fileName`	A filename. (Required) A filename can be absolute or relative to the current working directory or to any library on the library path. **Note**: You must specify a filename; other file objects that resolve to directories, libraries, cells, and views are not supported by the `dssIsFileLockedP` function. Likewise, you cannot specify the type of view object that DesignSync creates, for example: `~/ttlLib/and2/symbol.sync.cds`. These objects are not actual files; thus, you cannot apply the `dssIsFileLockedP` function to this type of object.
`g_vault`	Check whether the vault version of the file is locked (`t`), and, for module members, what branch of the vault object is locked. By default, the `dssIsFileLockedP` function checks whether the local workspace version of the file is locked (`nil`). **Note**: If you are checking for a lock on the vault version, only the current branch of the object is checked.

Value Returned

Returns the lock owner if the object is locked. **Note**: If you are checking for a lock in the local workspace, the lock owner returned is the owner of the object. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssIsViewLockedP

```
dssIsViewLockedP(
  t_libName t_cellName t_viewName [?vault g_vault]
)
=> t_user/nil
```

Description

Reports whether the specified cell view is locked, returning the lock owner if the cell view is locked or `nil` if the cell view is not locked. You can specify whether to check for a lock on the object in the local workspace or in the vault.

Arguments

`t_libName`	Library name. (Required)
`t_cellName`	Cell name. (Required)
`t_viewName`	View name. (Required)
`g_vault`	Check whether the vault version of the cell view is locked (`t`), and, for module members, what branch of the vault object is locked. By default, the `dssIsViewLockedP` function checks whether the local workspace version of the cell view is locked (`nil`). **Note**: If you are checking for a lock on the vault version, only the current branch of the object is checked.

Value Returned

Returns the lock owner if the object is locked. **Note**: If you are checking for a lock in the local workspace, the lock owner returned is the owner of the object. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssJoinLibraryP

```
dssJoinLibraryP(
  t_vaultPath [?libName t_libName] [?libPath t_libPath]
  [?mirrorPath t_mirrorPath] [?mode t_mode]
  [?selector t_selector] [?recursive g_recursive]
  [?retain g_retain] [?silent g_silent]
  [?background g_background]
)
=> t/nil
```

Description

Accesses the specified library and sets up an associated workspace.

Arguments

`t_vaultPath` Vault URL for the data repository:

`sync://<host>:<port>/Projects/<path>/<libraryName>`
(for SyncServer vaults)

`file:///<clientvaultpath>` (for client vaults)

	(Required) **Note:** You cannot specify a module vault URL or a workspace path to a module.
`t_libName`	Library name. By default, the library name is the last element of the vault path in the `t_vaultPath` argument.
`t_libPath`	Local path to the library, including the library name. The library path defaults to `<syncJoinLibDefaultPath>/<lib_name>`, where `<lib_name>` is the library name as specified in the `t_vaultPath` or `t_libName` argument. The default value for `<syncJoinLibDefaultPath>` is "`.`" where "`.`" is the DFII current working directory.
`t_mirrorPath`	Path of the local mirror directory, for example, `/home/karen/mirrors/ASIC`. By default, no mirror is set.
`t_mode`	Check-out mode ("`lock`", "`share`", "`mirror`", or "`get`"). By default, the mode matches the default fetch mode. See the DesignSync Data Manager DFII User's Guide:Selecting a Default Fetch Mode to learn how to set the default fetch mode.
`t_selector`	Selector specifying the branch of the library you want to access. For non-branching environments, specify the `Trunk` selector. By default, the selector of the parent directory of `t_libPath` is used.
`g_recursive`	Recurse to fetch entire library (`t`). (Default). To fetch just the library-level files, set to `nil`.
`g_retain`	Retain the "last modified" timestamps of the checked-out objects as recorded when the object was checked into the vault (`t`), or make the timestamps the check-out time (`nil`). The default is `nil` unless defined otherwise from SyncAdmin (see SyncAdmin Help). The retain option is only meaningful when checking out physical copies (`lock` and `get` modes) and is silently ignored otherwise.
`g_silent`	Run silently (`t`). (Default)
`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.

Value Returned

Returns `t` if the workspace associated with the specified library has been set up successfully; otherwise, returns `nil`.

dssLibraryStatusP

```
dssLibraryStatusP(
  t_libName [?silent g_silent]
)
=> l_status
```

Description

Reports library status, such as the workspace path. If the library is managed by DesignSync DFII, reports the vault path, module, selector, cache, and mirror information associated with the library, as well as SyncServer availability.

Arguments

`t_libName`	Library name. (Required)
`g_silent`	Run silently (`t`). (Default) To generate a descriptive status output similar to the Library Status form, set to `nil`.

Value Returned

Returns a disembodied property list of status information for the specified library.

`l_status`	Returns a list of the form `(nil`
	`name t_name`
	`modulePath t_modulePath`
	`libName t_libName`
	`libPath t_libPath`
	`dmType t_dmType`
	`vaultPath t_vaultPath`
	`cachePath t_cachePath`
	`selector t_selector`
	`localMirrorPath t_localMirrorPath`
	`vaultMirrorPath t_vaultMirrorPath`
	`serverStatus t_serverStatus)`
	
	If a value is unavailable for any reason, the value is `nil`.

`t_name`	The name of the module.
`t_modulepath`	The workspace instance of the module.
`t_libName`	The name of the library.
`t_libPath`	The path to the library's local workspace.

`t_dmType`	The design management tool associated with the library as defined by the Cadence `DMTYPE` variable. The value is `"sync"` if the library is managed by DesignSync DFII.
`t_vaultPath`	The vault folder associated with the managed library, or `nil` if the library is unmanaged.
`t_cachePath`	Path of the cache directory, as specified when DesignSync DFII was installed or from SyncAdmin.
`t_selector`	Selector specifying the branch or version of the library.
`t_localMirrorPath`	Path of the local mirror directory.
`t_vaultMirrorPath`	This legacy property's value, which pertains to old `"setvaultmirror"` functionality, is always `"nil"`.
`t_serverStatus`	Status of the SyncServer on which the library's vault resides. The value is `"up"` if the server is available or the vault is a client vault, and `"down"` if it is unavailable.

The function raises an error if argument checking fails.

Example

The following example shows the return format of the `dssLibraryStatusP` function.

```
dssLibraryStatusP("df2test")
=>
(nil
   localMirrorPath
"file:///home/tbarbg10/Mirrors/Libraries/df2test"
   vaultMirrorPath nil
   cachePath "/home/tbarbg10/Caches/Libraries/df2test"
   serverStatus "up"
   selector "Trunk"
   vaultPath "sync://qewfsun9:30138/Libs/df2test"
   dmType "sync"
   libPath "/home/tbarbg4/Cadence/df2test"
   libName "df2test"
)
```

The following example shows the output returned by the `dssLibraryStatusP` function with the `?silent` option set to `nil`.

```
dssLibraryStatusP("df2test" ?silent nil)
=>
   Library:          df2test
   Path:             /home/tbarbg4/Cadence/df2test
   DM Type:          sync
   Vault:            sync://qewfsun9:30138/Libs/df2test
   Cache Directory: /home/tbarbg10/Caches/Libraries/df2test
   Selector:        Trunk
   Mirror:         file:///home/tbarbg10/Mirrors/Libraries/df2test
   Server:          Accessible
   (nil localMirrorPath
"file:///home/tbarbg10/Mirrors/Libraries/df2test"
vaultMirrorPath nil cachePath
"/home/tbarbg10/Caches/Libraries/df2test"
serverStatus "up" selector "Trunk" vaultPath
"sync://qewfsun9:30138/Libs/df2test" dmType "sync" libPath
"/home/tbarbg4/Cadence/df2test" libName "df2test"
   )
```

dssListHierarchyP

```
dssListHierarchyP(
  t_libName t_cellName tl_viewNames
  [?switchUsing t_switchUsing]
  [?switchList l_switchList]
  [?stopList l_stopList]
  [?switchLibChoice S_switchLibChoice]
  [?switchLibNames l_switchLibNames]
  [?processViews gl_processViews]
  [?processFiles gS_processFiles] [?silent g_silent]
)
=> l_result
```

Description

Lists the objects of a design hierarchy in the workspace. To identify the cells in a design hierarchy, DesignSync DFII scans the hierarchy, beginning with the top-level cell views you specify using the `tl_viewNames` argument. Then, DesignSync DFII descends into the views indicated by the `t_switchUsing` argument. You can use the `t_switchUsing` argument to specify that DesignSync DFII descend into one or more views you specify in a switch list (using the `l_switchList` argument). You can instead have DesignSync DFII descend into all instantiated views or all views that exist for a cell by setting the `t_switchUsing` argument to `"instantiatedView"` or `"allViews"`, respectively. Use the `l_stopList` argument to indicate at which views DesignSync DFII is to stop scanning. DesignSync DFII also offers other hierarchy controls, such as limiting which libraries are scanned using the `S_switchLibChoice` argument and limiting which views are checked in using the `g_processViews` argument.

Notes:

- For DesignSync DFII to scan the hierarchy, the cells must be in your local workspace.
- DesignSync DFII does not scan through libraries that have been filtered out using the `l_switchLibNames` argument. For example, suppose a cell in **library_1** references a cell in **library_2**, which references a cell in **library_3**. If **library_2** is filtered out in the `l_switchLibNames` argument, the cell in **library_3** is not found.

Arguments

`t_libName`	Top library name of hierarchy to be listed. (Required)
`t_cellName`	Top cell name of hierarchy to be listed. (Required)
`tl_viewNames`	Top-level view names of hierarchies to be checked out. (Required) Can be given a single view, a string, or a list of views.
`t_switchUsing`	Indicates how the design hierarchy is to be traversed. Specify one of the following: • `"firstSwitchList"`: As the design is traversed, DesignSync DFII descends into the first view specified in the switch list that exists for a cell. Specify the switch list using the `l_switchList` argument. (Default) • `"allSwitchList"`: As the design is traversed, DesignSync DFII descends into each view of the cell in the workspace that matches a view in the switch list. Specify the switch list using the `l_switchList` argument. • `"instantiatedView"`: As the design is traversed, DesignSync DFII descends into each instantiated view. The `l_switchList` argument is ignored in this case. • `"allViews"`: As the design is traversed, DesignSync DFII descends into each view of the cell in the workspace that exists for each cell. The `l_switchList` argument is ignored in this case.
`l_switchList`	Names of the views to be scanned to identify the design hierarchy. The `l_switchList` argument is required if you specify the `"firstSwitchList"` or `"allSwitchList"` values using the `t_switchUsing` argument. If the `t_switchUsing` argument is set to `"instantiatedView"` or `"allViews"`, this argument is ignored.
`l_stopList`	Names of views at which the hierarchy scanning should

stop. As the design is traversed, if the `l_switchList` view being scanned is also in this list, scanning stops.

`S_switchLibChoice` Specifies which libraries to enter as the hierarchy is scanned:

- `all`: Enter all libraries. (Default)
- `only`: Enter only the libraries specified by the l`_switchLibNames` argument.
- `not`: Enter all libraries except those specified by the l`_switchLibNames` argument.

`l_switchLibNames` Library names controlled by the `S_switchLibChoice` argument. You need not include this argument if `all` is selected as the `S_switchLibChoice` argument.

`g_processViews` Once you have identified the hierarchy using the `t_switchUsing` argument, as well as the switch list and stop list if necessary, specify the views of the identified cells to be processed:

- `t`: Process all views that exist for the cell.
- `nil`: Process only the single view switched into. (Default)
- List of views to process.

`gS_processFiles` Specifies whether cell- and library-level files are processed in addition to the specified cell views:

- `nil`: No cell- or library-level files are processed. (Default)
- `cell`: Cell-level files are processed, but library-level files are not. This option selects only cell-level files for those cells on which you are operating.
- `library`: Cell- and library-level files are processed.

`g_silent` Run silently and provide no warning if no objects match (`t`). (Default). If you set `g_silent` to `nil`, warning messages are provided if no objects match.

Value Returned

Returns the list of objects in the specified design hierarchy. Each entry in the return list is a list containing an object's library name, its cell name, its cell view name, and its filename. The filename can be `nil`, indicating that the object is a cell view. The cell view can be `nil`, indicating a file at the cell level. Both the cell and cell view can be `nil`, indicating a file at the library level.

dssTagCategoryP

```
dssTagCategoryP(
  t_libName tl_catNames t_tag [?viewNames l_viewNames]
  [?move g_move] [?remove g_remove] [?nested g_nested]
  [?modified g_modified] [?silent g_silent]
  [?background g_background]
)
=> nil/(x_pass x_fail)
```

Description

Tags (or removes a tag from) objects of one or more categories. You can tag all the objects of a category at one time or specify views to tag.

Arguments

`t_libName`	Library name. (Required)
`tl_catNames`	One or more category names. (Required)
`t_tag`	Tag to apply to the workspace versions of the objects of the specified category. See DesignSync DFII Help for tag naming guidelines. (Required)
`l_viewNames`	One or more view name(s) to be tagged. (Optional). Tags all views in the workspace by default.
`g_move`	Move a tag that is already used on a version of an object to a new version (`t`). By default (`nil`), a tag operation fails if the tag is already in use, because a tag can be attached to only one version or branch of an object at a time. **Note**: The `g_move` and `g_remove` arguments are mutually exclusive.
`g_remove`	Delete the `t_tag` selector from the specified objects (`t`). By default (`nil`), the `t_tag` selector is added to the specified versions. **Note**: The `g_move` and `g_remove` arguments are mutually exclusive.
`g_nested`	Apply to nested category contents (`t`). (Default) **Note:** If `g_nested` is set to `t` but one or more nested category files are missing from your workspace, DesignSync DFII automatically fetches the missing category files and processes the specified objects.
`g_modified`	For locally modified objects, tag originally checked-out version (`t`) instead of failing (`nil`, default). Tagging is a vault operation,

	so locally modified objects are themselves never tagged. **Note**: The `g_remove` and `g_modified` arguments are mutually exclusive.
`g_silent`	Run silently (`t`). (Default)
`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.

Value Returned

Returns a list of pass and fail counts; the first integer represents the number of objects successfully tagged and the second integer represents the number of failures. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssTagCellP

```
dssTagCellP(
  t_libName tl_cellNames t_tag [?viewNames l_viewNames]
  [?move g_move] [?remove g_remove] [?silent g_silent]
  [?modified g_modified] [?background g_background]
)
=> nil/(x_pass x_fail)
```

Description

Tags (or removes a tag from) a cell in the workspace. To tag an entire design hierarchy, use the dssTagHierarchyP function.

Arguments

`t_libName`	Library name of cell to be tagged. (Required)
`tl_cellNames`	Names of one or more cells to be tagged. (Required)
`t_tag`	Tag to apply to the workspace versions of the objects. See the DesignSync Data Manager DFII User's Guide for tag naming guidelines. (Required)
`l_viewNames`	One or more view name(s) to be tagged. (Optional). Tags all views in the workspace by default.

`g_move`	Move a tag that is already used on a version of an object to a new version (`t`). By default (`nil`), a tag operation fails if the tag is already in use, because a tag can be attached to only one version or branch of an object at a time. **Note**: The `g_move` and `g_remove` arguments are mutually exclusive.
`g_remove`	Delete the `t_tag` selector from the specified objects (`t`). By default (`nil`), the `t_tag` selector is added to the specified versions. **Note**: The `g_move` and `g_remove` arguments are mutually exclusive.
`g_modified`	For locally modified objects, tag originally checked-out version (`t`) instead of failing (`nil`, default). Tagging is a vault operation, so locally modified objects are themselves never tagged. **Note**: The `g_remove` and `g_modified` arguments are mutually exclusive.
`g_background`	Run silently (`t`). (Default)
`g_silent`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.

Value Returned

Returns a list of pass and fail counts; the first integer represents the number of objects successfully tagged and the second integer represents the number of failures. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssTagCellViewP

```
dssTagCellViewP(
  t_libName t_cellName t_viewName t_tag
  [?versionName t_versionName] [?move g_move]
  [?remove g_remove] [?silent g_silent]
  [?modified g_modified] [?background g_background]
)
=> nil/(x_pass x_fail)
```

Description

Tags (or removes a tag from) a cell view in the workspace. To tag an entire design hierarchy, use the `dssTagHierarchyP` function.

Arguments

`t_libName`	Library name of cell view to be tagged. (Required)
`t_cellName`	Cell name of cell view to be tagged. (Required)
`t_viewName`	Cell view name to be tagged. (Required)
`t_tag`	Tag to apply to the workspace version of the object. See the DesignSync Data Manager DFII User's Guide for tag naming guidelines. (Required)
`t_versionName`	Version to be tagged. By default, the version is the current version in the workspace.
`g_move`	Move a tag that is already used on a version of an object to a new version (`t`). By default (`nil`), a tag operation fails if the tag is already in use, because a tag can be attached to only one version or branch of an object at a time. **Note**: The `g_move` and `g_remove` arguments are mutually exclusive.
`g_remove`	Delete the `t_tag` selector from the specified objects (`t`). By default (`nil`), the `t_tag` selector is added to the specified versions. **Note**: The `g_move` and `g_remove` arguments are mutually exclusive.
`g_modified`	For locally modified objects, tag originally checked-out version (`t`) instead of failing (`nil`, default). Tagging is a vault operation, so locally modified objects are themselves never tagged. **Note**: The `g_remove` and `g_modified` arguments are mutually exclusive.
`g_silent`	Run silently (`t`). (Default)
`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.

Value Returned

Returns a list of pass and fail counts; the first integer represents the number of objects successfully tagged and the second integer represents the number of failures. The `dssTagCellViewP` function lets you tag a single cell view only, so the returned list is (1 0) if the tag operation is successful and (0 1) if the tag operation fails. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssTagFileP

```
dssTagFileP(
  tl_fileNames t_tag [?move g_move] [?remove g_remove]
  [?modified g_modified] [?silent g_silent]
  [?background g_background]
)
=> nil/(x_pass x_fail)
```

Description

Tags (or removes a tag from) one or more file object(s) in the local workspace.

You can specify absolute or relative filenames. Filenames can be relative to the current working directory or to any library on the library path. For example, if library `acc` is on your library path, then you can specify the `cdsinfo.tag` file for that library as `acc/cdsinfo.tag`, even though the `acc` library directory might be anywhere on disk. If a library name exists, and there is also a directory within the current working directory of the same name, the library name is used.

Specify wildcards for filenames using glob-style expressions.

Notes:

For wildcards, filenames in the current working directory take precedence over library names. That is, a glob expression of `lib*` will not match libraries `libA` and `libB` if similarly named files exist in the current working directory; the `dssTagFileP` function first expands regular expressions against the current directory, and then performs library matching.

Arguments

`tl_fileNames`	One or more file object(s) to be tagged. (Required) You can specify file objects as glob-style expressions. A file object can be: A filename, specified as a full path or a path relative to the current working directory. A filename, specified relative to a library, for example `<libname>/cdsinfo.tag` or

	`<libname>/cellname/prop.xx`. A directory name, either a full path or a path relative to the current working directory. A library name. A cell name, specified as `<libname>/<cellname>`. A view name, specified as `<libname>/<cellname>/<viewname>`. **Note**: DesignSync creates objects called `<name>.sync.cds` to represent Cadence views, where `<name>` corresponds to the name of the view folder, for example: `~/ttlLib/and2/symbol.sync.cds`. These objects are not actual files; thus, you cannot apply the `dssTagFileP` function to this type of object.
`t_tag`	Tag to apply to the workspace versions of the objects. See the DesignSync Data Manager DFII User's Guide for tag naming guidelines. (Required)
`g_move`	Move a tag that is already used on a version of an object to a new version (`t`). By default (`nil`), a tag operation fails if the tag is already in use, because a tag can be attached to only one version or branch of an object at a time. **Note**: The `g_move` and `g_remove` arguments are mutually exclusive.
`g_remove`	Delete the `t_tag` selector from the specified objects (`t`). By default (`nil`), the `t_tag` selector is added to the specified versions. **Note**: The `g_move` and `g_remove` arguments are mutually exclusive.
`g_modified`	For locally modified objects, tag originally checked-out version (`t`) instead of failing (`nil`, default). Tagging is a vault operation, so locally modified objects are themselves never tagged. **Note**: The `g_remove` and `g_modified` arguments are mutually exclusive.
`g_silent`	Run silently (`t`). (Default)
`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of

background commands.

Value Returned

Returns a list of pass and fail counts; the first integer represents the number of objects successfully tagged and the second integer represents the number of failures. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssTagHierarchyP

```
dssTagHierarchyP(
  t_libName t_cellName tl_viewNames
  t_tag [?switchUsing t_switchUsing]
  [?switchList l_switchList] [?move g_move]
  [?remove g_remove] [?stopList l_stopList]
  [?switchLibChoice S_switchLibChoice]
  [?switchLibNames l_switchLibNames]
  [?processViews gl_processViews]
  [?processFiles gS_processFiles]
  [?modified g_modified] [?silent g_silent]
  [?background g_background]
)
=> nil/(x_pass x_fail)
```

Description

Tags (or removes a tag from) a design hierarchy in the workspace. To identify the cells in a design hierarchy, DesignSync DFII scans the hierarchy, beginning with the top-level cell views you specify using the `tl_viewNames` argument. Then, DesignSync DFII descends into the views indicated by the `t_switchUsing` argument. You can use the `t_switchUsing` argument to specify that DesignSync DFII descend into one or more views you specify in a switch list (using the `l_switchList` argument). You can instead have DesignSync DFII descend into all instantiated views or all views that exist for a cell by setting the `t_switchUsing` argument to `"instantiatedView"` or `"allViews"`, respectively. Use the `l_stopList` argument to indicate at which views DesignSync DFII is to stop scanning. DesignSync DFII also offers other hierarchy controls, such as limiting which libraries are scanned using the `S_switchLibChoice` argument and limiting which views are checked in using the `g_processViews` argument.

Notes:

- For DesignSync DFII to scan the hierarchy, the cells must be in your local workspace.

- DesignSync DFII does not scan through libraries that have been filtered out using the `l_switchLibNames` argument. For example, suppose a cell in **library_1** references a cell in **library_2**, which references a cell in **library_3**. If **library_2** is filtered out in the `l_switchLibNames` argument, the cell in **library_3** is not found.

Arguments

`t_libName`	Top library name of hierarchy to be tagged. (Required)
`t_cellName`	Top cell name of hierarchy to be tagged. (Required)
`tl_viewNames`	Top-level view names of hierarchies to be checked out. (Required) Can be given a single view, a string, or a list of views.
`t_tag`	Tag to apply to workspace versions of the design hierarchy. See the DesignSync Data Manager DFII User's Guide for tag naming guidelines. (Required)
`t_switchUsing`	Indicates how the design hierarchy is to be traversed. Specify one of the following: • `"firstSwitchList"`: As the design is traversed, DesignSync DFII descends into the first view specified in the switch list that exists for a cell. Specify the switch list using the `l_switchList` argument. (Default) • `"allSwitchList"`: As the design is traversed, DesignSync DFII descends into each view of the cell in the workspace that matches a view in the switch list. Specify the switch list using the `l_switchList` argument. • `"instantiatedView"`: As the design is traversed, DesignSync DFII descends into each instantiated view. The `l_switchList` argument is ignored in this case. • `"allViews"`: As the design is traversed, DesignSync DFII descends into each view of the cell in the workspace that exists for each cell. The `l_switchList` argument is ignored in this case.
`l_switchList`	Names of the views to be scanned to identify the design hierarchy. The `l_switchList` argument is required if you specify the `"firstSwitchList"` or `"allSwitchList"` values using the `t_switchUsing` argument. If the `t_switchUsing` argument is set to `"instantiatedView"` or `"allViews"`, this argument is ignored.

`g_move`	Move a tag that is already used on a version of an object to a new version (`t`). By default (`nil`), a tag operation fails if the tag is already in use, because a tag can be attached to only one version or branch of an object at a time. **Note**: The `g_move` and `g_remove` arguments are mutually exclusive.
`g_remove`	Delete the `t_tag` selector from the specified objects (`t`). By default (`nil`), the `t_tag` selector is added to the specified versions. **Note**: The `g_move` and `g_remove` arguments are mutually exclusive.
`l_stopList`	Names of views at which the hierarchy scanning should stop. As the design is traversed, if the `l_switchList` view being scanned is also in this list, scanning stops.
`S_switchLibChoice`	Specifies which libraries to enter as the hierarchy is scanned: • `all`: Enter all libraries. (Default) • `only`: Enter only the libraries specified by the l`_switchLibNames` argument. • `not`: Enter all libraries except those specified by the l`_switchLibNames` argument.
`l_switchLibNames`	Library names controlled by the `S_switchLibChoice` argument. You need not include this argument if `all` is selected as the `S_switchLibChoice` argument.
`g_processViews`	Once you have identified the hierarchy using the `t_switchUsing` argument, as well as the switch list and stop list if necessary, specify the views of the identified cells to be processed: • `t`: Process all views that exist for the cell. • `nil`: Process only the single view switched into. (Default) • List of views to process.
`gS_processFiles`	Specifies whether cell- and library-level files are processed in addition to the specified cell views: • `nil`: No cell- or library-level files are processed. (Default) • `cell`: Cell-level files are processed, but library-level files are not. This option selects only cell-level files for those cells on which you are operating. • `library`: Cell- and library-level files are processed.

`g_modified`	For locally modified objects, tag originally checked-out version (`t`) instead of failing (`nil`, default). Tagging is a vault operation, so locally modified objects are themselves never tagged. **Note**: The `g_remove` and `g_modified` arguments are mutually exclusive.
`g_silent`	Run silently (`t`). (Default)
`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.

Value Returned

Returns a list of pass and fail counts; the first integer represents the number of objects successfully tagged and the second integer represents the number of failures. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssTagLibraryP

```
dssTagLibraryP(
  t_libName t_tag [?viewNames l_viewNames]
  [?move g_move] [?remove g_remove]
  [?modified g_modified][?silent g_silent]
  [?background g_background]
)
=> nil/(x_pass x_fail)
```

Description

Tags (or removes a tag from) a library in the workspace.

Arguments

`t_libName`	Name of library to be tagged. (Required)
`t_tag`	Tag to apply to the workspace versions of the objects. See the DesignSync Data Manager DFII User's Guide for tag naming guidelines. (Required)

`l_viewNames`	One or more view name(s) to be tagged. (Optional). Tags all views in the workspace by default.
`g_move`	Move a tag that is already used on a version of an object to a new version (`t`). By default (`nil`), a tag operation fails if the tag is already in use, because a tag can be attached to only one version or branch of an object at a time. **Note**: The `g_move` and `g_remove` arguments are mutually exclusive.
`g_remove`	Delete the `t_tag` selector from the specified objects (`t`). By default (`nil`), the `t_tag` selector is added to the specified versions. **Note**: The `g_move` and `g_remove` arguments are mutually exclusive.
`g_modified`	For locally modified objects, tag originally checked-out version (`t`) instead of failing (`nil`, default). Tagging is a vault operation, so locally modified objects are themselves never tagged. **Note**: The `g_remove` and `g_modified` arguments are mutually exclusive.
`g_silent`	Run silently (`t`). (Default)
`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.

Value Returned

Returns a list of pass and fail counts; the first integer represents the number of objects successfully tagged and the second integer represents the number of failures. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssUnlockCellViewP

```
dssUnlockCellViewP(
  t_libName t_cellName t_viewName [?branch t_branch]
  [?silent g_silent] [?background g_background]
)
=> nil/(x_pass x_fail)
```

Description

Unlocks a single cell view. Use `dssCancelCellViewP` to remove the lock on a cell view that you have checked out in your workspace.

Arguments

`t_libName`	Library name. (Required)
`t_cellName`	Cell name. (Required)
`t_viewName`	View name. (Required)
`t_branch`	Branch name. By default, `dssUnlockCellViewP` unlocks the current branch of the cell view in your workspace. If the cell view is not in your workspace, then by default the library's selector is used.
`g_silent`	Run silently (`t`). (Default)
`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.

Value Returned

Returns a list of pass and fail counts; the first integer represents the number of successful unlocks and the second integer represents the number of failures. The `dssUnlockCellViewP` function lets you unlock a single cell view, so the returned list is (1 0) if the unlock is successful and (0 1) if the unlock fails. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssUnlockFileP

```
dssUnlockFileP(
  t_fileName [?branch t_branch] [?silent g_silent]
  [?background g_background]
)
=> nil/(x_pass x_fail)
```

Description

Unlocks a single file. Use `dssCancelFileP` to remove the lock on a file or module that you have checked out in your workspace.

You can specify an absolute or relative filename. Filenames can be relative to the current working directory or to any library on the library path. For example, if library `acc` is on your library path, then you can specify the `cdsinfo.tag` file for that library as `acc/cdsinfo.tag`, even though the `acc` library directory might be anywhere on disk. If a library name exists, and there is also a directory within the current working directory of the same name, the library name is used.

Arguments

`t_fileName`	A file or module name. (Required) A filename can be absolute or relative to the current working directory or to any library on the library path. **Note**: You must specify a filename; other file objects that resolve to directories, libraries, cells, and views are not supported by the `dssUnlockFileP` function. Likewise, you cannot specify the type of view object that DesignSync creates, for example: `~/ttlLib/and2/symbol.sync.cds`. These objects are not actual files; thus, you cannot apply the `dssUnlockFileP` function to this type of object.
`t_branch`	Branch name. By default, `dssUnlockFileP` unlocks the current branch of the file in your workspace. If the file is not in your workspace, then by default the selector of the library or parent folder is used.
`g_silent`	Run silently (`t`). (Default)
`g_background`	Run command in the background (`t`). By default, commands run in the foreground (`nil`). DesignSync DFII adds background commands to the Background Queue. Use the graphical interface command, **Synchronicity => Options => Show Background Queue** to view the queue. See Error Handling and Diagnostics: Return Values and Background Commands for information about the output of background commands.

Value Returned

Returns a list of pass and fail counts; the first integer represents the number of successful unlocks and the second integer represents the number of failures. The `dssUnlockFileP` function lets you unlock a single file, so the returned list is (1 0) if the unlock is successful and (0 1) if the unlock fails. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssViewDataSheetP

```
dssViewDataSheetP(
  t_fileName | t_libName [t_cellName] [t_viewName]
)
=> t/nil
```

Description

Displays the data sheet (in your HTML browser) for the specified object (library, cell, view, file, or directory).

Arguments

`t_fileName`	File or module name. (Required unless you specify a library name.) Module objects can be specified by full path. DesignSync objects can be specified as a full path or a path relative to the current working directory or library, for library, for example `<libname>/cdsinfo.tag` or `<libname>/<cellname>/prop.xx`. **Note**: • DesignSync creates objects called `<name>.sync.cds` to represent Cadence views, where `<name>` corresponds to the name of the view folder, for example: `~/ttlLib/and2/symbol.sync.cds`. These objects are not actual files; thus, you cannot apply the `dssViewDataSheet` function to this type of object. • You can also specify a directory to view the data sheet for that directory.
`t_libName`	Library name. (Required unless you specify a file name: `t_fileName`.)
`t_cellName`	Cell name.
`t_viewName`	View name.

Value Returned

Returns `t` if the data sheet can be displayed, otherwise, returns `nil`. The function raises an error if argument checking fails.

Example

The following examples show the invocation of the `dssViewDataSheetP` function.

In this example, the data sheet for a file is requested.

```
dssViewDataSheetP("/home/users/joe/libs/smallLib/cdsinfo.tag")
=> t
```

Because `smallLib` is a library defined in this user's `cds.lib` file, the following specification is equivalent:

```
dssViewDataSheetP("smallLib/cdsinfo.tag")
=> t
```

In this example, the data sheet for a cell view is requested:

```
dssViewDataSheetP("smallLib" "and2" "symbol")
=> t
```

dssViewVersionHistoryP

```
dssViewVersionHistoryP(
  t_fileName | t_libName [t_cellName [t_viewName]]?all g_all
 ?branch g_branch ?descendants g_descendants ?lastBranches
g_lastBranches ?lastVersions g_lastVersions ?maxTags g_maxTags
?report t_report?memberVault g_memberVault
)
=> t/nil
```

Description

Displays the version history (in a text window) for the specified file, module or cell view.

Arguments

`t_fileName`	File or workspace module name. (Required unless you specify a cell view: `t_libName t_cellName t_viewName`.) Module objects can be specified by full path. DesignSync objects can be specified as a full path or a path relative to the current working directory or library, for library, for example `<libname>/cdsinfo.tag` or

	`<libname>/<cellname>/prop.xx`. **Note**: DesignSync creates objects called `<name>.sync.cds` to represent Cadence views, where `<name>` corresponds to the name of the view folder, for example: `~/ttlLib/and2/symbol.sync.cds`. These objects are not actual files; thus, you cannot apply the `dssViewDataSheet` function to this type of object.
`t_libName`	Library name. (Required unless you specify `t_fileName`.)
`t_cellName`	Cell name. (Required unless you specify `t_fileName`.)
`t_viewName`	View name. (Required unless you specify `t_fileName`.)
`g_all`	Include all branches (`t`). Default is f, only include current branch.
`g_branch`	The branch name of the desired branch. Empty by default.
`t_descendants`	Number of descendant versions. This field is ignored when g_all is T.
`t_lastBranches`	Number of branches, from the current version back, to report on. Must be a positive integer.
`t_lastVersions`	Number of versions, from the current version back, to report on. Must be a positive integer.
`t_maxTags`	Maximum number of tags to report for a branch or version. Must be a positive integer.
`t_report`	Report mode keys. For a list of valid report mode keys and explanation of the keys, see *ENOVIA Synchronicity DesignSync DFII User's Guide:* Version History Report Options.
`g_memberVault`	t/nil boolean value indicating whether the command runs on the individual module member vault object or the the parent module. When the value is t, the command runs on the individual member vault (Default). When it is nil, the command runs on the parent module, providing a module history containing only the changes to the specified member. Note: When you specify a filename with the memberVault, you must include two sets of nil values between the filename value and the memberVault value.

Value Returned

Returns `t` if the version history can be displayed, otherwise, returns `nil`. The function raises an error if argument checking fails.

Example

The following examples show the invocation of the `dssViewVersionHistoryP` function.

In this example, the version history for a file is requested.

```
dssViewVersionHistoryP("/home/users/joe/libs/smallLib/cdsinfo.ta
g")
=> t
```

Because `smallLib` is a library defined in this user's `cds.lib` file, the following specification is equivalent:

```
dssViewVersionHistoryP("smallLib/cdsinfo.tag")
=> t
```

In this example, the version history for a cell view is requested:

```
dssViewVersionHistoryP("smallLib" "and2" "symbol")
=> t
```

Menu Customization Functions

Customizing the Synchronicity Menu

You (or your project leader) can configure the Synchronicity menu, selectively removing, adding, or reordering submenus and commands. For example, if your team uses scripts to place libraries under revision control, your project leader might remove **Configure Library** from the Synchronicity menu. Or if your team policy is never to delete cell view versions from the vault, you can remove **Delete => Version**.

You can control your menu configurations based on user level, whether the menu is in the CIW or cell view (also known as design editor or DE) window, and for both short and full versions of the cell view menu (see the DesignSync Data Manager DFII User's Guide: Controlling the Synchronicity Menu on Cell View Windows).

This section describes the SKILL functions used to configure the Synchronicity menu. These functions are defined when `dssInit.il` is loaded (see the DesignSync Data Manager DFII User's Guide: Loading the DesignSync Integration into DFII).

dssMenuAddItemP

```
dssMenuAddItemP(
  t_menu tl_items t_relative [?post g_post]
)
=> t/error
```

Description

Adds a menu item or submenu to a menu. If a menu item to be added is not already defined, you must first define it using the `dssMenuAddValidItemP` function, then use the `dssMenuAddItemP` function to specify where to place an instance of the new menu item within the existing menu structure.

Arguments

`t_menu`	The name of a menu to which the specified menu item or submenu is to be added (`CIWNovice`, `CIWAdvanced`, `CIWExpert`, `DENoviceFull`, `DEAdvancedFull`, `DEExpertFull`, `DENoviceShort`, `DEAdvancedShort` or `DEExpertShort`). (Required)
`tl_items`	Either the name of a predefined menu item or a list containing a new submenu name followed by its predefined menu items. (Required) **Note**: The predefined menu items can be menu items that already exist in the DesignSync DFII interface or you

	can define them using the `dssMenuAddValidItemP` function.
`t_relative`	The name of an existing menu item or submenu on the specified menu next to which the new item or submenu is to be added. To view the structure of the menu in order to choose where to insert the new item, use the `dssMenuListMenuP` function. (Use the `g_post` argument to specify whether the new item is to be added before or after the existing item.) (Required)
`g_post`	By default, the item is placed before the relative menu item or submenu (`nil`). Specify `t` to place the item after the relative menu item or submenu (`t_relative`). If `t_relative` is a submenu, then the new item is placed before or after that entire menu and not as an item on that menu.

Value Returned

Returns `t` if the item or submenu is successfully added. Raises an error if the menu or the relative item is not found.

Example

The following example adds the **Delete Version** menu item to the short version of the DE menu in expert mode only, after the existing **Delete** item:

```
dssMenuAddItemP("DEExpertShort" "DeleteVersion" "Delete" ?post
t)

syncUseEditWindowShortMenu = t

dssMenuRefreshP()
```

The first line adds the menu item. The second line turns on the use of the short menu on DE windows. The last line refreshes the displayed menus.

dssMenuAddValidItemP

```
dssMenuAddValidItemP(
  t_item r_initItem [?uninitItem r_uninitItem]
)
=> t
```

Description

Adds a new entry to the list of allowed menu items. This new menu item is specified as an existing SKILL menu item type. You can use the SKILL `hiCreateMenuItem` function to create a new type of SKILL menu item or use an existing SKILL menu item. See the Cadence SKILL documentation for more information about SKILL menu items.

If the menu item you are defining already exists, it is redefined with the SKILL menu items you specify using the `r_initItem` and `r_uninitItem` arguments.

Arguments

`t_item`	The name of the new menu item to be defined. (Required)
`r_initItem`	The SKILL menu item used to specify the new menu item. This menu item is used if the resulting form is to be initialized with the details of the current DE window cell view. (Required)
`r_uninitItem`	The SKILL menu item to be used if the resulting form is **not** to be initialized with the details of the current DE window cell view. The `r_uninitItem` argument is optional; include it only if there are circumstances when a form is not to be initialized. If the `r_uninitItem` argument is not specified, then the `r_initItem` menu item is always used. If you create a SKILL menu item using the SKILL `hiCreateMenuItem` function, ensure that the menu callbacks are designed to appropriately take advantage of this feature.

Value Returned

Returns `t`.

Example

The following example shows how to create a new menu item and use the item within a Synchronicity menu. The `hiCreateMenuItem` SKILL function creates a SKILL menu item, `myMenuItem`, which prints the current time. Next, the `dssMenuAddValidItemP` function creates a new menu item, `Time`, defined as a `myMenuItem` SKILL menu item. Finally, the `dssMenuAddItemP` function adds the new menu item, `Time`, to the advanced mode of the CIW menu, before the existing **Options** item.

```
myMenuItem = hiCreateMenuItem(?name 'myMenuItem ?itemText "Time"
?callback "println(getCurrentTime())")

dssMenuAddValidItemP("Time" myMenuItem)

dssMenuAddItemP("CIWAdvanced" "Time" "Options")

dssMenuRefreshP()
```

dssMenuListItemsP

```
dssMenuListItemsP(
  [?print g_print]
)
=> l_names
```

Description

Returns, and optionally prints, the names of all valid menu items.

Arguments

`g_print`	Specifies whether the names should be printed, as well as returned, by the function call (`t`). By default, the names are only returned by the function and not printed (`nil`).

Value Returned

`l_names`	A list of all the valid menu items.

dssMenuListMenuP

```
dssMenuListMenuP(
  [?menu t_menu] [?port p_port]
)
=> t
```

Description

Prints the existing structure for one or all of the menus. The menu structure includes the name of each menu item and the submenus within it. Each submenu level is indented and preceded by the submenu name.

There are nine different menus stored, depending on the user level, whether the menu is in the CIW or DE window, and whether the menu is in the short or full version of the DE window.

Arguments

`t_menu`	The name of a menu (`CIWNovice`, `CIWAdvanced`, `CIWExpert`, `DENoviceFull`, `DEAdvancedFull`, `DEExpertFull`, `DENoviceShort`, `DEAdvancedShort` or `DEExpertShort`).

	To view the structure of all menus, specify `all`. (Default)
`p_port`	A SKILL port to which the output is to be written. The default port is `poport`.

Value Returned

Returns `t`.

dssMenuLoadConfigP

```
dssMenuLoadConfigP(
  t_fileName
)
=> t/error
```

Description

Load the menu configuration from the specified file. This is a simple alias for the SKILL `load` function; the menu configuration file is stored in a SKILL executable format. In addition to the menu configuration saved by the `dssMenuSaveConfigP` function, the SKILL file can also contain custom menu item definitions.

Arguments

`t_filename`	Name of the file from which to load the menu structure and any additional custom menu definitions. (Required)

Value Returned

Returns `t` if the menu configuration is successfully loaded. Raises an error if the file cannot be opened.

dssMenuRefreshP

```
dssMenuRefreshP()
=> t
```

Description

Refreshes the menus attached to all windows using the currently stored menu definitions. Call this function after making any changes to the menu structures. Call this function also after changing variables that control whether the long or short menus are used and whether CIW callbacks initialize the forms.

Arguments

None.

Value Returned

Returns `t`.

dssMenuRemoveItemAllP

```
dssMenuRemoveItemAllP(
  t_item
)
=> t
```

Description

Removes a menu item or submenu from all menus where it is currently used.

Arguments

`t_item`	The name of a menu item or submenu name. (Required)

Value Returned

Returns `t`.

dssMenuRemoveItemP

```
dssMenuRemoveItemP(
  t_menu t_item [?silent g_silent]
)
=> t/error
```

Description

Removes a menu item or submenu from a menu. The first matching item or submenu (using a depth-first search) is removed. To remove an item or submenu from all menus, use the `dssMenuRemoveItemAllP` function.

Arguments

`t_menu`	The name of a menu from which you are removing the specified menu item (`CIWNovice`, `CIWAdvanced`, `CIWExpert`,

	`DENoviceFull`, `DEAdvancedFull`, `DEExpertFull`, `DENoviceShort`, `DEAdvancedShort` or `DEExpertShort`). (Required)
`t_item`	The name of a menu item or submenu name to be removed. (Required)
`g_silent`	Run silently (`t`). (Default)

Value Returned

Returns `t` if the item or submenu is successfully removed. Raises an error if the specified item is not currently on the menu.

Example

The following example removes the **Delete => Version** item from the CIW and DE menus in novice mode:

```
dssMenuRemoveItemP("CIWNovice" "Delete->Version")

dssMenuRemoveItemP("DENoviceFull" "Delete->Version")

dssMenuRefreshP()
```

The first two lines remove the menu item from the menus. The last line refreshes the displayed menus.

dssMenuRemoveValidItemP

```
dssMenuRemoveValidItemP(
  t_item
)
=> t
```

Description

Removes the specified item from the list of allowed menu items. Note that if the specified item is currently used in the menu structures, warnings are generated when the menus are refreshed.

Arguments

`t_item`	The name of the existing menu item to be removed from the list

of valid menu items. (Required)

Value Returned

Returns `t`.

dssMenuSaveConfigP

```
dssMenuSaveConfigP(
  t_fileName
)
=> t/error
```

Description

Saves the current menu structures in the specified file. Note that the list of valid menu items is **not** saved, as it contains references to SKILL menu items, which cannot be stored.

This function is intended as a simple way of storing a menu structure once it has been created. If you use custom menu items, then you also need to store the commands required to create those menu items and add them to the list of valid items. However, you can store these additional commands in the same file used to save the menu structure (specified with the `t_filename` argument) because the structure of the file is a SKILL source code file. To load the menu structure and any additional custom menu item definitions, use the `dssMenuLoadConfigP` function, which calls the SKILL `load` function.

Arguments

`t_filename`	Name of the file in which to store the menu structure. (Required)

Value Returned

Returns `t` if the menu configuration is successfully saved. Raises an error if the file cannot be opened.

dssRefreshWindowBannerP

```
dssRefreshWindowBannerP(
  [?windows rl_windows]
)
=> t/nil
```

Description

Refreshes the window banner for one or more windows.

Arguments

`rl_windows`	The identifier of a window or a list of windows to refresh. The default is to refresh all windows. To list all the windows that currently exist, use the `hiGetWindowList()`SKILL function.

Value Returned

Returns `t` if the windows have been successfully refreshed; otherwise, returns `nil`.

Miscellaneous Functions

dssChangeDefaultsContextP

```
dssChangeDefaultsContextP(
  [?context t_context]
)
=> t_context
```

Description

Changes the context of default values that are saved, when using Save Defaults in a DesignSync DFII form. See the DesignSync Data Manager DFII User's Guide: Setting Form Default Values. Default values can be saved by an individual user, for a project team, or for all users of a site's software installation. Similarly, the context determines which default values are removed, either for a user, a project team, or the entire site. See the DesignSync Data Manager DFII User's Guide: Viewing and Resetting Form Defaults.

By default, the saving and removing of default values apply only to the user who invoked DesignSync DFII.

If you are a project leader, and set your context to `project,` your saving and removing of default values will apply to all members of the project team. See DesignSync Data Manager Administrator's Guide: Project-Specific Configuration. Individual users can override default values that were set for their project team.

If you are the site administrator, and set your context to `site`, your saving and removing of default values will apply to all users of the software installation. See DesignSync Data Manager Administrator's Guide: Site-Wide Configuration. Project leaders can override default values that were set site-wide.

Invoke the `dssChangeDefaultsContextP` function with no argument to determine the current context.

Arguments

`t_context`	The context to change to (`user`, `project`, or `site`). (Optional) If not specified, the context remains unchanged.

Value Returned

`t_context`	Returns the updated context if the `t_context` argument was

	specified; otherwise, returns the existing context.

dssChangeUserLevelP

```
dssChangeUserLevelP(
  [?level t_level]
)
=> t_level
```

Description

Changes the user level for the current user to the level specified. Invoke the `dssChangeUserLevelP` function with no argument to determine the current user level. See DesignSync DFII Help: Selecting a User Level for a description of the user levels.

Arguments

`t_level`	User level to change to (`novice`, `advanced`, or `expert`). (Optional) If not specified, level remains unchanged.

Value Returned

`t_level`	Returns the updated user level if the `t_level` argument was specified; otherwise, returns the existing user level.

dssEnableDebugP

```
dssEnableDebugP(
  [?enable g_enable]
)
=> t/nil
```

Description

Turns debug mode on or off. With debug mode enabled, DesignSync DFII generates output in the CIW, displaying all commands sent to the stclc process and the output from those commands.

Arguments

`g_enable`	Enable debugging (`t`). Default. To turn off debugging, set `g_enable` to '`nil`'.

Value Returned

Returns `t` if debug mode is successfully enabled (`g_enable` set to '`t`'). Returns `nil` if debug mode is successfully turned off (`g_enable` set to '`nil`').

dssExecuteTclP

```
dssExecuteTclP(
  t_cmd [?print g_print] [g_args]...
)
=> l_result
```

Description

Executes a Tcl command in the stclc process used by DesignSync DFII. You can choose whether to print the output of the command and return values to the CIW.

Note: DesignSync commands invoked via dssExecuteTclP do not use the command line defaults system. The command line defaults system only pertains to DesignSync command line shells.

Arguments

`t_cmd`	Tcl command to be executed. (Required) This argument can be a format string as used for the `printf` function.
`g_print`	Print the output and return values to the CIW (`t`). By default, the output is not printed to the CIW.
`g_args`	Any additional values required by the `t_cmd` argument.

Value Returned

`l_result`	List of strings, one for each line of output, including return values.

If the `g_print` argument is set to '`t`', the output and return values display in the CIW.

Example

The following example calls the DesignSync `synctrace` command to turn on command tracing. See the **ENOVIA Synchronicity Command Reference** for more information about the `synctrace` command.

```
dssExecuteTclP("synctrace set 0")
```

dssHelpP

```
dssHelpP()
=> t
```

Description

Displays the the DesignSync Data Manager DFII User's Guide main page in a web browser.

Arguments

None.

Value Returned

Returns `t`.

dssSetWorkspaceRootPathP

```
dssSetWorkspaceRootPathP(

 tl_path

)
=> t/nil
```

Description

Sets the workspace root path. The workspace root path is a list of paths to modules that are not in the current working directory for the Cadence client or known library directories.

Arguments

`tl_path`	List of directory paths.

Value Returned

Returns `t` if the workspace root path is set successfully, otherwise, returns `nil`.

Example

The following example sets the workspace root paths.

```
dssSetWorkspaceRootPathP()
```

dssGetWorkspaceRootPathP

```
dssGetWorkspaceRootPathP()
=> l_path
```

Description

Returns the workspace root path. The workspace root path is a list of paths to workspace module roots that are not in the current working directory for the Cadence client or known library directories.

Arguments

Value Returned

`l_path`	List of directory paths.

Example

The following example lists the workspace root paths.

```
dssGetWorkspaceRootPathP()

=> ("/home/rsmith/cadenceworkspaces/" "/home/rsmith/MyMods/")
```

Getting Assistance

Using Help

ENOVIA Synchronicity Product Documentation provides information you need to use the products effectively. The Online Help is delivered through WebHelp ® , an HTML-based format.

Note:

Use SyncAdmin to change your default Web browser, as specified during ENOVIA Synchronicity tools installation.

To open the **DesignSync Data Manager DFII SKILL Programming Interface Guide**, do one of the following:

- Enter the following URL from your Web browser:

  ```
  http://<host>:<port>/syncinc/doc/dsdfiiref/dsdfiiref.htm
  ```

  where <host> and <port> are the SyncServer host and port information. Use this server-based invocation when you are not on the same local area network (LAN) as the DesignSync installation.

- Enter the following URL from your Web browser:

  ```
  file:///$SYNC_DIR/share/content/doc/dsdfiiref/dsdfiiref.htm
  ```

  where $SYNC_DIR is the location of the DesignSync installation. Specify the value of SYNC_DIR, not the variable itself. Use this invocation when you are on the same LAN as the installation. This local invocation may be faster than the server-based invocation, does not tie up a server process, and can be used even when the SyncServer is unavailable.

When the Online Help is open, you can find information in several ways:

- Use the **Contents** tab to see the help topics organized hierarchically.
- Use the **Index** tab to access the keyword index.
- Use the **Search** tab to perform a full-text search.

Within each topic, there are the following navigation buttons:

- **Show** and **Hide**: Clicking these buttons toggles the display of the navigation (left) pane of WebHelp, which contains the Contents, Index, and Search tabs. Hiding the navigation pane gives more screen real estate to the displayed topic.

Showing the navigation pane gives you access to the Contents, Index, and Search navigation tools.

- **<<** and **>>**: Clicking these buttons pages to the previous or next topic in the help system.

You can also use your browser navigation aids, such as the **Back** and **Forward** buttons, to navigate the help system.

Related Topics

Getting a Printable Version of Help

Getting a Printable Version of Help

The *DesignSync Data Manager DFII SKILL Programming Interface Guide* is available in book format from the ENOVIA Documentation CD or the DSDocumentationPortal_Server installation available on TheLink. The content of the book is identical to that of the help system. Use the book format when you want to print the documentation, otherwise the help format is recommended so you can take advantage of the extensive hyperlinks available in the DesignSync Help.

You must have Adobe® Acrobat® Reader™ Version 8 or later installed to view the documentation. You can download Acrobat Reader from the Adobe web site.

Related Topics

Using Help

Accessing Product Documentation

You can access the complete set DesignSync and ProjectSync documentation through the ENOVIA Synchronicity Product Documentation page.

To access the Product Documentation page, click ENOVIA Synchronicity Product Documentation.

To access the Product Documentation page from outside this help system, enter one of the following URLs from your Web browser:

```
file:///<SYNC_DIR>/share/content/doc/index.html
```

-or-

```
http://<host>:<port>/syncinc/doc/index.html
```

Contacting ENOVIA

- For solutions to technical problems, please use the 3ds web-based support system:
    - http://media.3ds.com/support/

- From the 3ds support website, you can access the Knowledge Base, General Issues, Closed Issues, New Product Features and Enhancements, and Q&A's. If you are not able to solve your problem using this information, you can submit a Service Request (SR) that will be answered by an ENOVIA Synchronicity Support Engineer.

- If you are not a registered user of the 3ds support site, send email to ENOVIA Customer Support requesting an account for product support:

- enovia.matrixone.help@3ds.com

Related Topics

Using Help

DesignSync Glossary

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A

acadmin

A command-line interface for managing access controls. DesignSync also provides a graphical Web-based interface, Access_Administrator.

accelerator key

An accelerator key is a key associated with a menu choice. Pressing the accelerator key is acts as a shortcut to selecting the menu choice. For example, pressing **F7** has the same effect as selecting **Revision Control => Check In**.

Access Administrator

A web-based interface for managing access controls graphically. DesignSync also provides a command-line interface, **`acadmin`**.

access control

A means of protecting design objects so that only an authorized user or user group can perform operations on objects. The access control commands also let you limit the operations and design objects these users can access. See also the Access Control Guide.

active selection

At any given time, both the tree view and list view can contain a set of selected objects. However, only one of these sets is the active selection. Most menu choices operate an the active selection. The view that does not contain the active selection indicates this by using a faded color to highlight selected objects. You can make either the tree view or the list view the active selection by clicking on the view or setting the keyboard focus to that view.

add

An operation to designate an object not currently under revision control as a member of a module.

administrator

A person who sets up an ENOVIA Synchronicity tool, for example DesignSync or ProjectSync for use by others. The set-up can include defining access_controls, note types, command defaults, client defaults etc.

aggregate view data

The candidate set of module members that results when you apply more than one module view to a workspace. DesignSync uses the **initial view data** from each view to build an aggregate view which is then populated. Filters and hreffilters can then be applied to the aggregate view data which results in the data populated into the workspace.

Apache

A public domain web server. The widely used Apache server is developed and maintained by the Apache Server Project. See the Apache web site, www.apache.org, for more information.

authentication

Verification performed by a software product to ensure that a particular user has the right to use the product or components of the product. DesignSync provides the capability of protecting your design data and preventing unauthorized users from accessing your data through the access control commands. See also DesignSync Data Manager User's Guide: Accessing a SyncServer with User Authentication, Access Control Guide: User Authentication Access Controls.

auto branching

A type of branching in which a new branch is created automatically if one does not already exist as part of a checkin or checkout with lock. Auto-branching useful when a developer wants to explore a "what if" scenario.

B

base directory

See module base directory.

bookmark

A placeholder you create in DesignSync to let you quickly visit a working directory, file, or vault you access often. See also DesignSync Data Manager User's Guide: Bookmarks.

branch

A thread of development of a managed design object that diverges from the main development thread of the object. The main branch (Trunk) is designated with branch number 1 and files on this branch have version numbers 1.1, 1.2, 1.3, and so on. The branch number always has an odd number of digits and the version number for a design object always has an even number of digits. For example, if you create the first branch from version 1.2 of a design object, the branch is designated 1.2.1 (a second branch would be designated 1.2.2) and the first version on that branch is 1.2.1.1. In this example, version 1.2 of the design object is referred to as the branch point version. See also DesignSync Data Manager User's Guide: Creating Branches.

branch-point version

A design object version that is the root of a new branch. For example, if you create a new branch off version 1.2 of a design object, that version is the branch-point version. If this is the third branch off this version, it is designated 1.2.3, and the first version on the new branch is 1.2.3.1. See also DesignSync Data Manager User's Guide: Creating Branches.

branch tag

A tag can be applied to either a branch or an object version. A tag applied to a branch is called a branch tag. See also tag, immutable, mutable, version tag, DesignSync Data Manager User's Guide: Tagging Versions and Branches.

C

cache

- An area on a LAN (local area network) where common files used by a team can be stored to avoid redundant local copies. A cache gives users their own view of the design data, while minimizing disk utilization. DesignSync caches are implemented using UNIX hard links and symbolic links, depending on the system setup. For clarity, the documentation refers to these types of caches as file caches or LAN caches. See also LAN,

  There is no support on Windows for cached files.

  Caches are not intended for use with modules. For information on cached modules, see Module Cache.

- The directory that DesignSync uses for temporary files during operations such as comparing file versions.

cancel

To undo a checkout. Cancel removes the locks on specified checked-out objects.

category

A module category is a virtual path to a module within the module area. All modules are stored in the Modules area. Categories provide a means to separate and organize the modules into related groups. Categories are created implicitly when you specify a module path in at module creation time.

cell

The fundamental element of Milkyway design data on which users operate. A cell always contains at least one file with a name that has the form <cell_name>:<local_version>, for example, `route:1`. A cell may contain attached files, whose names have the form <cell_name>:<local_version>_<number> (where number is any integer). For example, `route:1_2`.

In the file system, the files that make up a cell are located in a view directory. See also view.

cell view collection

A collection object that represents a Milkyway cell in a Milkyway view directory. DesignSync displays this object as cell.sync.mw, where cell is the cell name, for example, `route.sync.mw`. A cell view collection includes a cell view file and additional files called cell view attachment files.

Cell View File

A member of the DS MW cell view collection. This file is the primary cell file. The file has a name that has the form <cell_name>:<local_version>, for example, `route:1`.

cell view attachment file

A member of the DS MW cell view collection. This file is an attached file of the cell, a file whose name has the form <cell_name>:<local_version>_<number> (where number is any integer). For example, `route:1_2`.

cell view non-member

A DS MW cell view file or cell view attachment file that is part of a local version that is not the highest local version of the cell in the workspace. For example, suppose a cell named `route` has local versions `route:1` and `route:2`. The file `route:2` is a cell view file but the file `route:1` is a cell view nonmember. Nonmembers are not part of the current cell collection.

A cell view non-member can also be a cell view attachment file that is no longer attached to the cell. Such a non-member can occur due to an error situation in the

Synopsys Milkyway tools. DesignSync does not consider these "unattached" attached files to be part of a collection; DesignSync identifies them as non-versionable and does not check them into the vault.

cell view lock

A file created by the Milkyway tools. This file is a temporary file and cannot be checked in to the DesignSync vault.

cell view tmp file

A file created by the Milkyway tools. This file is a temporary file and cannot be checked in to the DesignSync vault.

checkin

Process of storing a snapshot of a design object in a vault. The new snapshot is called a version. See DesignSync Data Manager User's Guide: Checking in Design Files.

checkout

Process of making a revision-controlled design object available in your local area. You can check out a locked or unlocked copy of the object, a link to a cache or mirror copy of the object, or a reference to the object. See also populate, DesignSync Data Manager User's Guide: Checking out Design Files.

client

A process communicating with a server process in order to use a service provided by the server. The service provided by the SyncServer process is the management of design objects under revision control. There are 5 DesignSync client applications: DesSync (DesignSync GUI), dss (DesignSync Shell), dssc (concurrent dss), stcl (Synchronicity Tcl), and stclc (concurrent stcl). There are also integrations (or add-ons) into clients to support management of design objects through the software used to create and edit those objects, for example DSDFII provides access to Cadence data, and DSVS provides access to Visual Studio projects. See also server, DesSync, dss, dssc, stcl, stclc, DesignSync Data Manager User's Guide: DesignSync Command-Line Shells.

client defaults

Most of the graphical forms used to run commands in the graphical clients, for example DSDFII, DSVS, and DesSync, allow you to save the options settings as the default value for the commands. When the command form is run again, the form preselects the saved defaults. If you do not wish to use those defaults, you can manually select different settings. These client defaults are entirely independent of each other and the command defaults allowing you to finely customize your environment.

client trigger

A trigger that is executed on the client. See also trigger, DesignSync Data Manager Administrator's Guide: Triggers Overview.

collection object

A group of files that together define a design object and are revision controlled as a single object. For example, a schematic may consist of dozens or even hundreds of files within any number of folders. You operate on the design object as a single entity while letting your design tools manage the object at the file level. See also DesignSync Data Manager User's Guide: Collections Overview.

command defaults

Default values can be set for the options specified to commands run from the command line. Default values can be set for individual commands, command families (such as all "access" sub-commands), or all commands. This simplifies invocations of commands from the command line, because the user does not need to specify "-[option] <value>" for the saved default value. Command defaults can be set by the administrator for all users on the server, or by individuals for their own commands. See also client defaults, Command Reference: command defaults.

components

Files or other objects that made up a project's hierarchy.

Physical - When using DesignSync, physical components are the files and directories created automatically during the normal process of checking in a design.

Logical - A logical component is an artificial project category set up for the explicit purpose of distinguishing where notes should be attached.

compression

Conversion of a data representation into an equivalent but smaller representation requiring fewer bytes. Compression optimizes storage space and speeds up transmission of data. DesignSync compresses data in its vaults. See also DesignSync Data Manager Administrator’s Guide:Turning Off Compression.

configuration

See configuration management.

configuration management

Associated design objects that together form a snapshot of a design project. DesignSync provides two methods of associating design objects: grouping the objects together into a module, or applying a common tag to the design objects. Modules can also be tagged and associated with other modules using tags or hierarchical references.

copy out

See fetch.

CVS

Concurrent Versions System. A software tool for UNIX systems that manages multiple versions of files, tracking changes and controlling shared files. CVS uses a merging work style where a design team does not lock file versions, but instead merges their edits with the latest checked-in version. See also merging work style.

D

data sheet

The information about an object under DesignSync revision control displayed in an HTML browser or the DesSync client. The information displayed depends on the type of object. For example, the data sheet for a file in your working folder contains its lock status, modification status, version number, and associated tags. See also DesignSync Data Manager User's Guide: What Are Data Sheets?.

DES

Data Encryption Standard. A highly effective data encryption algorithm developed at IBM in 1977. DES uses 72 quadrillion (72,000,000,000,000,000) encryption keys, chosen at random during encryption. Communication between DesignSync and SyncServers uses the DES encryption algorithm for encryption. See also encryption.

DesignSync web interface

The DesignSync web interface, built into ProjectSync, provides a method of configuring your DesignSync environment and users through a Web-page based interface.

design configuration

See configuration_management.

designer role

A user who contributes to the development of block or modules in a SITaR environment.

DesSync

The DesignSync graphical user interface (GUI), one of the DesignSync client applications. To invoke the GUI, enter "DesSync" from your operating-system window (UNIX or Windows). See also client, dssc, stclc, Command Reference: DesSync command description.

domain name

A unique alphanumeric identifier for a computer resource on the Internet. A domain name is a meaningful descriptor for a web resource used in lieu of its IP address, a string of digits. An example of a domain name is host.yourcompany.com. See also IP address, DNS.

DNS

Domain Name System. A system for maintaining a distributed list of Internet domain names and their corresponding IP addresses. DNS locates the domain names and translates these into IP addresses accessible by computers on the Internet. See also Internet, domain name, and IP address.

DSDFII

ENOVIA Synchronicity DesignSync® DFII(TM) is the integration of many DesignSync® design-management (DM) capabilities into the Cadence Design Systems DFII environment. See also DesignSync Data Manager DFII User’s Guide

DS MW

ENOVIA Synchronicity DesignSync® MW(TM) (DS MW) is the integration of many DesignSync design management capabilities into the environment of Milkyway-based tools. A Milkyway-based tool reads and writes to a set of files that conform to the Milkyway database, as defined by Synopsys. DesignSync MW provides features to manage large, geographically dispersed projects. See also DesignSync Data Manager MW User’s Guide.

dss

The DesignSync shell, one of the DesignSync client applications. To invoke dss, enter "dss" from your operating-system window (UNIX or Windows). You can also enter "dss" followed by a DesignSync command to execute a single command. In most cases, use dssc (concurrent dss) instead of dss to take advantage of the concurrent client capabilities. The dss client uses syncd to communicate with a SyncServer. See also dssc, client, Command Reference: dss command description, DesignSync Data Manager's User's Guide: DesignSync Command-Line Shells.

dssc

The concurrent version of the DesignSync shell (dss). The dssc client has several advantages over dss, because it's designed to be used concurrently with other client processes. To invoke dssc, enter "dssc" from your operating-system window (UNIX or Windows), optionally specifying a single DesignSync command to execute. The dssc client does not use syncd to communicate with a SyncServer. See also dss, client, Command Reference: dssc command description, DesignSync Data Manager User's Guide: DesignSync Command-Line Shells.

DSVS

ENOVIA Synchronicity DesignSync® VS) is the integration of many DesignSync® design-management (DM) capabilities into the Microsoft Visual Studio® environment. See also DesignSync Data Manager for Visual Studio User's Guide.

dynamic href mode

The selector associated with the hierarchical reference is always evaluated to identify the version of the sub-module to be operated on.

E

EDA

Electronic design automation, the tools and processes that facilitate the design of electronic systems.

encryption

Process of converting data into a secure format that impedes unauthorized use of the data. The data is then "decrypted" (or "deciphered") by the intended receiver of the data using a decryption key -- the algorithm that converts the data back to its original format. A "strong encryption" is an encryption method that is seemingly impossible to break without the use of the decryption key. DesignSync encrypts communications between DesignSync and the SyncServers to protect your design data from unauthorized use using the DES encryption algorithm. See also DES, DesignSync Data Manager Administrator's Guide: Overview of Secure Communications.

end user

A person who uses the system, but performs no system or site wide configuration or maintenance. The end-user benefits from the administrator's set-up. In a SITaR environment, both the designer role and integrator role are end-users.

external module

A link from the DesignSync change management system (CMS) to another CMS. The files within the external CMS are managed inside DesignSync as a module. This allows you to fetch a complete code base into a workspace, even when the files are managed by different change management application, and provides a common application interface for populate operations.

For more information on external modules, see DesignSync Data Manager System Administrator's Guide: Overview of External Modules.

F

fetch

To obtain a version of a design object without locking the object. You fetch an object by performing a populate, checkout, or get operation on objects without a lock. You can fetch an unlocked copy of the object, a link to a cache or mirror copy of the object, or a reference to the object. The integrations do not support fetching object references. See also DesignSync Data Manager System Administrator's Guide: Defining a Default Fetch State, DesignSync Data Manager User's Guide: Checking out Design Files.

file

A file is a collection of data stored in one unit, under a filename. This can be a document, a picture, an audio or video file, a library, an application, or other collection of data.

firewall

A set of programs running on an organization's gateway server that acts as a virtual barrier, protecting the organization's intranet from outside access. See also gateway server, intranet.

fixed tag

A logical tag construct applied once to a single snapshot of a set of object versions and never moved. This tag may be either mutable or immutable. A tag is considered "fixed" if you keep the tag associated with a particular snapshot rather than reapplying the tag to newer versions of the design objects. There are no special attributes for DesignSync tags to indicate whether they are fixed or movable. These terms are used to describe ways in which you can use design configuration tags. See also tag, movable tag, DesignSync Data Manager User's Guide: Using Tags.

folder

A file system directory.

G

gateway server

A computer on an organization's intranet through which all communications between the intranet and the WWW are transmitted. Firewall software runs on the gateway server, ensuring that only authorized communications occur. See also firewall, intranet,.

GDM

Generic Data Manager. A Cadence Design Systems API (application programming interface) used to integrate design management tools into the Cadence architecture. DSDFII follows the GDM standard and can therefore be used with GDM-based interfaces, such as Library Manager.

get

Term used for a populate or checkout operation in DSVS. See also fetch.

groupware

Collaborative management tool used by a group of people.

H

Hard link

A hard link (UNIX only) is an additional name for an existing file. Any number of hard links, and thus any number of names, can be created for any file. Hard links cannot cross file system boundaries or span across file partitions. The operating system does not distinguish between the original file name and any hard links that are subsequently created to that file; other than they are multiple names for the same file because both point to the same inode. The ownership of the hard link is defined by the original file, not the creator of the hard link.

Hard links and symbolic links are used to support the DesignSync file cache feature.

For information on cache usage, see the *ENOVIA System Administration Guide*: Introduction to Data Replication.

Hercules

A Milkyway-based tool that runs layout verification.

hierarchical reference

A dependant link from a module to any of the following: a module branch or version, a legacy module, a DesignSync vault, or an IP Gear deliverable. The hierarchical reference link allows you to build a hierarchical structure to model your project usage. The SITaR model is one example of how hierarchical references can be used to facilitate controlled code reuse.

href mode

See hierarchical_reference, normal href model ; static href mode, dynamic href mode.

http

The portion of a URL address that indicates the protocol used by the software to determine how to access an Internet resource. The "http" protocol supports such resources as HTML files, CGI applications, and Java applets. See also URL https.

https

The portion of a URL that indicates the protocol used by the software to determine how to access an internet resource. The "https" protocol supports Secure Sockets Layer (SSL) communication. See also encryption, URL.

I

immutable

An immutable object, usually a tag, that cannot change. For example, when a module is created, branch 1 is also created with an immutable tag of "Trunk". This tag is always associated with this branch. You can also add immutable tags to a version or branch when you want to uniquely and permanently identify the branch or version. An immutable tag can be used to designate a fixed tag. See also: mutable, movable tags, tag.

integrator role

A use who verifies the stability of blocks or modules, and assembles them into functional packages to provide a qualified and stable integrated code baseline in a SITaR workflow environment.

initial data

The module members that are populated into the workspace when there are no module views and no filter, exclude or hreffilters applied to the data.

initial view data

The contents of a single module view with no filters, excludes, or hreffilters applied to data. See also **aggregate view data**. If you load a single module view into the workspace, this is the set of candidate module members to populate. The data in the workspace may be further refined by the applying filters and hreffilters.

integrations

Collective term for software packages that have plug-in or add-on DesignSync capabilities, for example, DSDFII, or DSVS.

Internet

A world-wide system of computer networks initially developed by the Advanced Research Projects Agency (ARPA) of the US government. The Internet encompasses the networks and protocols (computer communication conventions) coordinating the transfer of data between computers.

intranet

A network of computers internal to a company or other organization protected against unauthorized access.

IP address

Internet protocol address. A unique identifier for the location of a computer resource on the Internet. An IP address is a string of digits, for example, 127.0.01. See also domain name, DNS.

IP Gear deliverable

A IP Gear deliverable is a package of data that has been uploaded and associated with an IP Gear Catalog Component.

J

K

keyword

Placeholder used as a comment in a revision-controlled object to express information about the design object, for example, the author ($Author$), check-in time ($Date$), or revision number ($Revision$). The values of keywords are expanded when you check in the object. See DesignSync Data Manager User's Guide: Revision Control Keywords Overview.

L

LAN

Local area network. A group of interconnected computers able to access data on other machines on the local network. Using DesignSync, a team can define an area on a LAN as a cache to share access to revision-controlled design objects. See also cache.

legacy module

A grouped set of DesignSync objects that represent one level of a design hierarchy. Legacies modules can be grouped into configurations, aliases, or releases, but the individual objects are managed independently. Legacy modules were obsoleted with version 5.0 which introduced a new type of module managed as a single entity in DesignSync. Legacy modules can still be included in a module hierarchy as a sub-module to a module. Legacy modules can also be upgraded to modules. See also hierarchical reference, DesignSync Data Manager User's Guide: Upgrading Legacy Modules.

library

A directory tree of Milkyway data. The library has the same name as the top directory. The top directory of a library always includes the library file, a binary file called `lib`.

library file

A binary file containing metadata about a Milkyway library. Each Milkyway library has a library file, called `lib`, which resides in the top directory of the library. The information in the library file includes:

- Details of the views/cells present in the library
- Library technology information
- A list of reference libraries
- Details of library attached files
- General library properties added by various tools
- Information on logical equivalence of cells

The library file is specific to the subset of cells that has been populated to a user's workspace. Two users might be collaborating on the same library, but each may have populated a different subset of cells to the workspace and therefore each may have a different library file. DS MW properly handles this file, allowing multiple users to collaborate on the same library.

library attachment

In Milkway, A file attached to a library. The file is named in the form `lib_X`.

library lock

The Milkyway lock file, which is a file named `.lock` that resides in the library directory. This file is a temporary file and cannot be checked in to the DesignSync vault.

library shadow collection

A collection object that encompasses the shadow files for the Milkyway lib file and its attached files. DesignSync displays this object as `shadow.sync.mw.lib`. A library shadow collection object contains the library file shadow and library attachment shadow files.

library file shadow

A member of the DS MW library shadow collection. This file contains a copy of the contents of the Milkyway library file (`lib`). The file has the name `shadow.lib`.

library attachment shadow

A member of the DS MW library shadow collection. This file contains a copy of the contents of the `lib_X` attached file. The file has the name `shadow.lib.X`.

An export Other Library Information operation copies the library file (lib) to the `shadow.lib` file and copies the library attached files (lib_X) to `shadow.lib.X`. Together these files form the library shadow collection object (`shadow.sync.mw.lib`). The export Other Library Information operation then checks in the library shadow collection object to the DesignSync vault.

When you import Other Library Information (using the Import Library Information form) DesignSync fetches the library shadow collection object from the vault to your workspace and loads the Other Library Information into your library file.

link

See symbolic link.

Local version

Multiple versions of a cell in Milkyway can exist in a library at the same time. Milkyway-based tools always operate on the version with the highest number in the library. This DS MW documentation refers to this as the local version, to distinguish it from a

DesignSync version, which is created in the DesignSync vault at check-in time (a vault version).

When a Milkyway object is checked in to the DesignSync vault, DesignSync applies a tag that indicates the object's local version number. If the local version is 6, DesignSync applies the tag `MW_6`.

lock

Attribute of a design object indicating that you or another team member is using and most likely editing the object. The owner of the locked object has the exclusive right to check it in to create the next version. See also locking work style.

locking work style

A team work style where each member locks an object while editing it, preventing other team members from checking in local modifications as new versions, until the locker checks in again. See also DesignSync Data Manager User's Guide: Locking or Merging Work Style?.

look & feel

The DesignSync GUI client allows you to customize the look and feel to determine the appearance and behavior of the UI. For example, the appearance of the tool bar, and the accelerator keys for the Cut, Copy, and Paste operations, depend on the look and feel. There are three look and feels available: Java (Metal), Motif, and Windows (not available on UNIX platforms).

M

merge

A merge allows you to combine your locally modified version of a design object with the Latest version of the object, resolving inconsistencies between the versions. In the merging work style, team members check out or populate without locking objects, thus, merging is required to reconcile the streams. Team members can "merge to" a branch to align the local metadata with the target branch, or "merge from" a branch to leave the metadata aligned with the current branch. See also merging work style.

merge edge

A merge edge indicates an extra "derived from" version for a version. The merge edge information is stored in the vault. When you perform a subsequent merge on the vault object, the merge edge is used to determine what changes have already been made, improving the efficiency of the merge.

merging work style

A team work style where team members check out objects without locking them. Multiple team members can edit the same object at one time; the first checked-in version becomes the Latest version and team members have to merge their edits in with this Latest version. See also DesignSync Data Manager User's Guide: Locking or Merging Work Style?.

metadata

Information about the objects that DesignSync manages, including the object's vault, branches (including current branch), versions and version-last-retrieved, its status (locked, unlocked, retired), its state as a local object (copy, locked copy, reference, link to cache or mirror directory), configuration tags applied to the object, and timestamps of operations on the object. Metadata is stored both on the server and locally. Local metadata is stored in `.SYNC` folders on your local file system. Note: DesignSync manages these .SYNC metadata folders; do not directly manipulate these folders or their contents. See also DesignSync Data Manager User's Guide: Metadata Overview.

Milkyway

The name of a database format, data model, and a directory structure that tools can adopt.

mirror directory

A directory that is automatically updated to contain the data set defined by a project leader for your project vault. For example, your team may always want access to the Latest version of files on the main Trunk branch. Another team may always want access to the file versions with a specific tag that are on a development branch. DesignSync creates links from users' work areas to the design objects in the mirror directory, ensuring that you have the most up-to-date configuration for your project.

A mirror directory saves team members from performing populate or check-out operations each time they want to access design changes made by team members. Because DesignSync mirrors are implemented using UNIX symbolic links, there is no support for mirrors on Windows. See DesignSync Data Manager Administrator's Guide: Mirroring Overview.

module

A module is a collection of managed objects that together make up a single entity. For example, DesignSync is a module composed of several sub-modules, such as ProjectSync and DSDFII, which contain all the code, examples, and documentation necessary to develop the applications.

The data included in a module includes its own objects and references to other modules, but not the contents of the referenced modules. See DesignSync Data Manager User's Guide: What is a Module? for more information.

module base directory

The top workspace directory of a module.

You specify the location of the base directory with the **populate** command using the **Working directory** option (`populate -dir`) when you get the module to your work area. (If you do not specify the **Working directory** option, the operation uses your current work area directory.)

module cache

A module cache is as a shareable workspace used for storing static modules to share with users across the network. Users link to the shareable workspace instead of to the module on the SyncServer. DesignSync Administrators or Project Managers can create module caches for modules containing common tools, libraries, or other modules needed for reference. Instead of populating the full contents of a module, UNIX users can populate a module cache link that provides access to the data without copying all the files locally. This reduces the populate time for the user as well as load on the server.

A project leader fetches modules into a module cache, preferably in "share" mode, to utilize DesignSync's file caching. When users populate module data, they can specify whether to link to modules in the module cache, or fetch modules from the server. How module data is retrieved is referred to as the module cache mode. The default module cache path, and the default module cache mode, can be defined in the registry by the team leader or by an individual user.

Note: Legacy modules can be linked to, or copied from, a module cache. The module cache mode of "copy" only applies to legacy modules. If you attempt to copy a non-legacy module from a module cache, DesignSync fetches the module from the server.

module cache link

A managed link to a module cache (mcache). The mcache link provides access to the module contents without requiring the user to load the contents of the module into her workspace by using UNIX symbolic links.

module category

A module category is a logical folder on the DesignSync server used to group modules into related groups.

For instance, tools developed and maintained for internal use, can be in a DesignSync category "Tools" which differentiate them immediately from applications being developed for external use.

module delta

Module delta mode shows only the files, folders, and hierarchical references that have been changed or added in the current version when you expand the module version on the server.

module member

A module member is any individual file, directory, or collection object added to a module.

module root

The module root of a module is the top level directory into which a module is fetched when users populate the module to their work areas.

module view

A module view is a filtered sub-group of module members. A module view definition is loaded onto the server so that the filtered sub-group is available to users who have access to the module but only need a specific set of files. For example, a project may contain source code, documentation, and compiled code. The development team may require only the source code. The release engineering team may require both the source code and the compiled code. The quality assurance team may require only the compiled code. The documentation team may require only the documentation. Using views, the team members do not have to populate the entire module or customize their own filtering at populate time.

module view definition

A module view definition is a TCL list that defines the module view. The module view definition is created in a local file on the system and then loaded or removed from the server with the `view` command set. The module view definition contains the name of the view, an optional description, and the filters and hreffilters that define the module view.

movable tag

A logical tag construct applied to a progression of snapshots of a set of object versions. A tag is considered "movable" if you reapply it to new versions of the objects as development progresses. A moveable tag must be mutable. If a tag is never meant to be moved, it is considered a fixed tag. See also tag, DesignSync Data Manager User's Guide: Fixed Tags and Movable Tags.

mutable

A mutable object has the ability to change. In DesignSync, this term is usually used in relationship to tags. For example, the creation of module branch, it is assigned an immutable module branch tag of Trunk which can not be altered. However, you can add mutable tags to versions of this module branch such as Test, Gold, Silver, Release, December. You can move these tags as needed. Mutable and immutable tags can be used to enforce logical tag constructs like movable tags and fixed tags. See also tag.

N

natural path

The natural path is the path where that object is placed under the module base directory.

normal href mode

The selector associated with the href is examined. If it represents a static version, which means that it is a version numeric, or is a version tag, or a list of such items, then that module is fetched using the selector and the mode switches to static for sub-modules. Otherwise, the module is still fetched using the selector, but the mode remains norma for subsequent levels of hierarchy.

netlist

See RTL Team, Place and Route.

note

A note is a record of information that you attach to a project or a project configuration. You can attach notes to multiple objects, including design files. Each note is of a particular note type with specific fields, or properties, you use to enter data for ProjectSync to track. See also note type, property.

note type

A note type is a database schema of fields (properties) for users to enter data for ProjectSync to track. ProjectSync provides standard note types for the ProjectSync server administrator to install. The administrator can modify the standard note types, as well as creating custom note types. See also note, property, property type.

O

object

A DesignSync object, commonly referred to as an "object' in the DesignSync documentation, is a file, folder, module, collection, symbolic link, or other object that can be managed by DesignSync.

other library information

Information contained in the Milkyway library file other than technology information, reference libraries, and the catalog of cells for the local directory. Examples of Other Library Information are:

- Logical equivalence of cells, such as those used by Astro during optimization
- Cosmos pcells
- Properties stored by Cosmos during schematic driven layout
- A reference control file

Note: The Other Library Information can include the technology file. If your DesignSync administrator has disabled the "Manage technology file separately" option on the SyncAdmin Third Party Integration form, DS MW manages the technology file as part of the Other Library Information. See DesignSync Data Manager MW User's Guide: Managing the Technology File for information.

overlapping modules

A workspace folder can be populated with files from more than one module. These modules are considered overlapping because a single workspace directory contains members of more than one module. When you have overlapping modules, you must explicitly add any new members to the correct module before checkin. You can check in or populate the modules either independently or in a single operation.

overlay

Fetching a file specified by the selector-list, and placing it in the working directory without changing the metadata. In other words, the user remains working on the same branch as before.

P

panel

A ProjectSync window or subwindow containing properties. Most panels behave like a graphical user interface (GUI) form element in which you enter data in fields and other GUI widgets. See also property.

persistent selector list

Specify what branch or version a command operates a version or branch is not explicitly specified. Checkin, checkout, populate, and import operations support persistent selector lists. Commands that do not use the persistent selector list typically operate on the current version or current branch of the object in your workspace. An object's persistent selector list is stored in local metadata or is inherited from the parent folder. See also selector, selector list.

place and route

One of the Milkyway-based tools (Astro) performs this step, where a netlist is used to describe which components need to be placed and which wires or "nets' need to be routed among the components.

populate

Process of making a revision-controlled design object available in your workspace. You can populate with locked or unlocked copies of objects, links to cache or mirror copies of objects, or references to objects. The objects populated reflect those objects existing in the vault and not those objects currently residing in your local area, unlike a recursive checkout where only the objects currently residing in your local area are checked out. Populate and checkout can be used interchangeably for all DesignSync objects except modules and module members which can only be populated into your workspace. See also DesignSync Data Manager User's Guide: Populating Your Work Area.

port number

Identifier for a specific server process through which a client process sends and receives data. The registered port number for SyncServer (Synchronicity server) processes is 2647.

project

A group of revision-controlled design objects stored together as a single design effort. You can manage caches on a per project basis -- in creating a project, you associate a vault with a cache to create a local view of the design data. See also DesignSync Data Manager's User's Guide: What Is a Project?, DesignSync Data Manager's User's Guide: Design Reuse.

ProjectSync

ENOVIA Synchronicity ProjectSync is a web-based system for managing engineering projects. ProjectSync helps engineering teams track bug reports, change orders, and continuing dialogue on engineering projects.

property

A property in ProjectSync is an element of a panel, like a field or other widget on a form in the graphical user interface (GUI). A property on a note type is characterized by 4 attributes - a name, a prompt string (optional), a type, and a default value (optional). See also note, note type, panel, ProjectSync User's Guide: What Are Properties?.

property type

A data type corresponding to a property on a note type. Examples of predefined property types include Boolean, Date, Integer, Float, and String. See also property, note type.

protocol

Portion of a URL address that indicates how to access the Internet resource. The "http" protocol, for example, supports such resources as HTML files, CGI applications, and Java applets. The DesignSync "sync" protocol is used to communicate with a SyncServer, while the "file" protocol can be used to navigate your local file system. See also DesignSync Data Manager User's Guide: DesignSync URLs.

proxy

A program residing on a firewall host that intercepts communications between a company's intranet and the Internet, only forwarding a request if the requesting party can be authenticated. Proxies can also perform tasks such as caching data to improve transmission speeds -- for example, the proxy might store a previously requested web page in cached memory to improve response time for subsequent requests for the data. DesignSync provides a registry setting and environment variable so that you can specify a proxy IP address and port number. See also authentication, cache, firewall, intranet.

prune

To remove unneeded versions from a vault to free up disk space. See DesignSync Help: Deleting Versions from a Vault.

Q

R

RCS

Revision Control System. A software tool for UNIX systems that manages multiple versions of files, tracking changes and controlling shared files. RCS uses a locking work style where a user locks a file version and thus has the exclusive right to check in the next version of that file. See locking work style.

reference

1. An object that is not physically present, but instead points to another object. The object has local metadata . When an object is checked in with the option to leave a reference to the object, the local copy is deleted, and a reference pointing to the vault replaces it. The integrations do not support references. See also Object States, regenerate.
2. `REFERENCE` also refers to a pointer you create in a ProjectSync project (in a `sync_project.txt` file); use REFERENCEs to import modules from other design projects into your design. See also DesignSync Data Manager User's Guide: Using Vault REFERENCEs for Design Reuse.
3. An ASCII file created and modified by DS MW to list the reference libraries for DesignSync maintenance.

   When you set up a library (using the Library Setup Options form) or export reference libraries (using the Export Library Information form), DS MW captures the path to each reference library in your design. For each reference library in your design, the `refs.txt` file contains a line with the structure: REFERENCE path_to_library, for example:

   REFERENCE /home/takeda/mde_lab/lab3/defoutlib

   The pathname_to_library can be an absolute or relative path. The pathname_to_library can also take the form `$MW_REFLIB/path`, where `MW_REFLIB` specifies an environment variable and path specifies a path relative to the directory defined by `$MW_REFLIB`.

   When you populate or when you check out a library to a new workspace, DesignSync fetches the `refs.txt` file from the vault to your workspace and loads the reference libraries the file identifies into your library file.

reference Library

A Milkyway cell in a library under DesignSync control may contain a reference to a cell in another library. The other library is called a reference library. In order to instantiate cells into your cell, you must first have declared the reference libraries with a Milkyway command, which causes a pointer to the reference library to be injected into the library file.

When you populate a new workspace, that operation cannot be completed until the library file in the new workspace is made aware of the reference libraries.

reference control file mode

Commonly, the library reference information for a Milkyway library resides in its `lib` file. In reference control file mode, library reference information resides in one of the library's attached files. This attached file is similar in form to the DS MW `refs.txt` file, except that it also contains the path to the library itself.

When a library is in reference control file mode, DS MW does not export the library references to a separate file. Instead, DS MW maintains the attached file as part of the Other Library Information.

To put a library into reference control file mode, use Scheme calls. See also DesignSync Data Manager MW User's Guide: Putting a Library into Reference Control Files Mode.

Note:

- When a library is in reference control file mode, the Reference Library field of the Library Setup and Export Library Information forms cannot be edited.
- If you import a library from the vault to a different library name in your workspace (for example, if you populate a library into a new workspace with a different name), DS MW cannot determine the name of the library and therefore cannot update the library reference information in the attached file. You must edit the attached file and correct the library references manually.

regenerate

DSDFII locked reference mode used when regenerating the object from a different source rather than editing it directly. This mode does not transfer the object from the vault. This is a more efficient mode for that use model since the user does not require the latest version of the generated file as long as the source files are correct. See also reference.

registry

The key-based repository used by DesignSync to store DesignSync settings. It is designed to mirror in functionality and layout the Windows Operating System registry. The registry supports customizing settings for the entire site, a specific server, a project, and an individual user. This allows a user to personalize the DesignSync environment. For more information, see registry files, DesignSync Data Manager Administrator's Guide: Overview of Registry Files.

registry file

A file containing setup and tool configuration information for DesignSync tools. DesignSync uses different registry files for different functions. Some of these registry files are only for use by the DesignSync administrator while others can be customized by individual users or project managers. See also registry.

relative path

The relative path indicates the path from the upper-level module to the object it is creating the connection to. The object can be a module, legacy module, configuration, IP Gear deliverable, or DesignSync vault.

remove

An operation to remove a module member from a module. This removes the member from all future operations on the module. You can add the object back to the module later, if needed.

replica

A copy of a design object created when you fetch (check out without a lock) a version from the vault. A replica contains relational information about the original replicated object, so that DesignSync can determine, for instance, whether the original object has changed since you fetched it. See also DesignSync Data Manager User's Guide: Object States.

repository

See vault.

retire

Prevents a design object from being automatically included in populate or recursive checkout operations. Retiring an object's branch rather than deleting the object itself gives you the opportunity to reinstate the object for future populate and recursive checkouts. Module objects are never retired, they are removed from the module. See also DesignSync Data Manager User's Guide: Retiring Branches.

RevisionControl note

A ProjectSync note created when a DesignSync revision control operation takes place. The note has a built-in type, RevisionControl, that contains information about the operation, such as the name of the user, command used to invoke the operation, and execution time. See also DesignSync Data Manager Administrator's Guide: RevisionControl Notes Overview.

root directory

See workspace root directory.

RSA

Rivest, Sharmir, and Adleman Internet encryption and authentication system. RSA is a commonly used encryption and authentication algorithm embedded in most web browsers. The algorithm uses separate keys -- a public key a sender uses for encrypting and a private key used by the owner to decrypt messages. The private key is never transmitted across the Internet, thus protecting the encryption from being deciphered. See also encryption.

RTL team

A group of people who create RTL source code, typically in Verilog or VHDL format. These people put the RTL code through synthesis to produce a netlist. A netlist serves as an entry point to Place and Route.

S

save settings

See client defaults.

SCC (Source Code Control)

DSVS features Windows-based integration that uses MSSCCI standard and is SCC compliant meaning that in additional to be integrated with Visual Studio, DSVS can be used with any other Windows application that supports an SCC plug-in.

selector

An expression that identifies a branch and version of a managed object. For example, the version tag "gold", the branch selector "Rel2:Latest", the version number "1.4", and the reserved keyword "Latest" are all selectors. Selectors are specified with -version and -branch options to various commands or through a persistent selector list. See also selector list, persistent selector list.

selector list

A comma-separated list of selectors. DesignSync uses the selector list as a search order to select a version on which to operate. If the first selector in the list results in no match, DesignSync retries the command using the next selector in the list. The command fails only if all selectors in the list produce no match. An example of a selector list is "gold,silver,Rel3.0:Latest,Trunk". See also selector, persistent selector list.

server

A computer process that provides services to client processes. The service provided by the SyncServer process is the management of the revision control and configuration tasks on the data in the DesignSync vaults. See also client.

server-side trigger

A trigger that runs on the server. See also trigger.

SITaR (Submit, Integrate, Test, and Release)

SITaR (Submit, Integrate, Test, and Release) is a simple module-based workflow for projects that consist of multiple blocks, or modules, developed by several contributors. SITaR uses hierarchical references to create a controlled integration environment for testing the interaction between the contents of modules in the hierarchy. See also, DesignSync Data Manager User's Guide: Overview of SITaR Workflow

skip

The action of skipping interim versions when checking in a new version of a design object which you did not lock, and replacing the Latest version. Skipping differs from a standard checkin in that interim versions can not be seen by team members checking out the Latest version; if you want the changes from the interim versions, you must request the versions explicitly by version number and enter their changes manually, or, if the file is not a binary file, you may be able to merge the changes into the file. Note: Skipping is an advanced operation and should be used with caution because it retires versions from the vault. You can only access the skipped versions if you enter their version numbers explicitly. See also DesignSync Data Manager User's Guide: Checking in Design Files.

SSL

A protocol to provide encrypted communications across a network. SSL is indicated by the https designated in a URL.

staging area

Intermediate folder used while relocating or converting vault folders (repositories). For example, if you move a vault folder to a new SyncServer, you first export the vault folder to an intermediate folder, then import the intermediate directory into a new SyncServer.

stcl

ENOVIA Synchronicity's customized Tcl (Tool Command Language) interpreter, one of the DesignSync client applications. The stcl client combines the text commands available in dss (the DesignSync shell) with the general scripting capabilities of Tcl. To invoke stcl, enter "stcl" from your operation-system window (UNIX or Windows). You can also enter "stcl" followed by a script name to execute an stcl script, or enter "stcl -exp" followed by a DesignSync or Tcl command in quotes to execute a single command. In most cases, use stclc to take advantage of the concurrent client capabilities. See also Command Reference: stcl command description, DesignSync Data Manager User's Guide: DesignSync Command-Line Shells.

stclc

The concurrent version of stcl. The stclc client has several advantages over stcl, because it's designed to be used concurrently with other client processes. To invoke stclc, enter "stclc" from your operation-system window (UNIX or Windows). You can also enter "stclc" followed by a script name to execute an stcl script, or enter "stclc -exp" followed by a DesignSync or Tcl command in quotes to execute a single command. See also Command Reference: stcl command description, DesignSync Data Manager User's Guide: DesignSync Command-Line Shells.

static href mode

Resolves hierarchical references to the static version of the sub-module recorded with the href at the time the hierarchical reference was created is always used. See also hierarchical_reference, dynamic reference.

substitution tag

ProjectSync extension to HTML that lets you embed placeholders for GUI widgets within a panel's HTML template. Substitution tags are coded as HTML comments in the template in the following format:

```
 <!-- SYNC <substname> -->
```

where `<substname>` is the substitution tag name. You can define the behavior for the substitution widget by modifying the installation (`.ini`) file corresponding to the panel's HTML template (or by creating an initialization file for the template if one does not exist.)

symbolic link

(UNIX symbolic link) A special UNIX file that points to another file. A link takes up less space than a copy of the original file. Use the UNIX **ln** command to create symbolic links. See the UNIX **ln** documentation for more information. See also DesignSync Data Manager User's Guide: Object States, DesignSync Data Manager Administrator's Guide: How DesignSync Handles Symbolic Links.

Symbolic links are also used with mirror directories , caches, and module caches. Also see hard link.

sync

The portion of a URL address that indicates the protocol used by the software to determine how to access an Internet resource. The "sync" protocol is used to communicate with a SyncServer. See also protocol, URL.

syncd

The Synchronicity daemon process. The syncd process manages communication between dss/stcl sessions and SyncServers. Note that dssc and stclc do not use syncd; they communicate directly with a SyncServer. See also Command Reference: syncdadmin.

SyncAdmin

The Synchronicity Administrator tool. LAN administrators can use SyncAdmin to set site-wide preferences, such as the default fetch preference, how symbolic links are managed, and which cache users should access. Users can use SyncAdmin to set user preferences such as their default editor and HTML browser. SyncAdmin is a standalone tool located in `<SYNC_DIR>/bin`. See also DesignSync Data Manager Administrator's Guide.

sync_project.txt file

A file created when you create a project in ProjectSync. The sync_project.txt file includes the REFERENCE and CONFIG mapping statements that let you import modules from other projects into your designs. See also ProjectSync User's Guide.

SyncServer

A DesignSync server process that manages shared information, controls access to design files, and performs administrative functions such as user privilege validation. A SyncServer is an http server, with a special "sync" protocol layered on the http protocol, for example, sync://host:port/Projects/Sportster. See also server, protocol, DesignSync Data Manager User's Guide: What Is a SyncServer?.

sync_servers.txt file

A file created by a user (My Servers), project leader (Site Servers), or administrator (Enterprise Servers) that lists SyncServer or vault definitions, which facilitates entering URLs in various locations in the DesignSync graphical interface and integrations; such as the DesignSync workspace wizard and the DSDFII Join Library wizard.. See DesignSync Data Manager Administrator's Guide: SyncServer List Files.

sync_server_list File

The DesignSync installation script, sync_install, automatically generates the sync_server_list file. Do not edit the sync_server_list file.

T

tag

An identifier you attach to a group of design object versions that you want to identify as a group. You can then perform operations on the group of objects as a whole. For example, you might tag all versions that went into a release "Rel2_0" so that at a later date another user could fetch all versions tagged "Rel2_0" thereby recreating that release. Tags can also be applied to branches. See also design configuration, mutable, immutable, DesignSync Data Manager User's Guide: Tagging Versions and Branches.

Tcl

Tool Command Language, a scripting language and interpreter developed at University of California, Berkeley by Dr. John Ousterhout. Tcl interpreters are embedded in many EDA applications, including DesignSync. See also stcl, stclc.

TCP/IP

Transmission Control Protocol/Internet Protocol, the communication conventions, or protocol, for transferring data on the Internet. Computers accessing the Internet have copies of the TCP/IP program which manages data transmissions, decoding addresses and transmitting the data to the specified IP address. See also IP address, domain name, DNS.

technology file

An ASCII file (`tech/tech.tf`) created and modified by DS MW with the purpose of making DesignSync aware of the library technology data for a design.

When you populate or when you check out a library to a new workspace, DesignSync fetches the `tech.tf` file from the vault to your workspace and loads the technology data the file identifies into your library file.

Note: If your DesignSync administrator has disabled the "Manage technology file separately" option on the SyncAdmin Third Party Integration form, DS MW manages the technology file as part of the Other Library Information. See also DesignSync Data Manager MW User's Guide: Managing the Technology File.

template

ProjectSync implements note panels with HTML templates. Administrators can generate an HTML template from an existing note type, then modify the HTML code to change its appearance.

trigger

A watchpoint (a Tcl script) that causes an action, such as sending an email, to occur automatically in response to some other action, such as checking in a file. A **client**

trigger is a trigger that is executed on the client. A server-side trigger is a trigger that runs on the server.

Trigger arguments

Trigger arguments are passed to any triggers that have been set up for the operation. Consult your project leader for information about any triggers that are in use and how they use arguments.

U

unique identifier

Every object and folder in the vault of a module is assigned a unique identifier. These unique identifiers are used within the module's versions. There is a mapping from the unique identifier to a natural path. The natural path can change from one module version to another, however, changing the natural path in this manner does not change the object's unique identifier.

unlock

Remove a lock from a design object's branch. You can unlock branches you have locked or those locked by team members. Access controls can be set up to limit unlocking operations. See also DesignSync Data Manager User's Guide: Undoing a Check Out, DesignSync Data Manager User's Guide: Unlocking Branches.

URL

Uniform resource locator, an address for a resource. A URL consists of the access protocol, followed optionally by a machine name and/or the path to the object. DesignSync uses a special "sync" protocol, for example, sync://host:port/Projects/Sportster. Use the "file" protocol to access files on a local network. See also http. https, sync, IP address, domain name.

user

Any individual authorized to work with DesignSync or ProjectSync.

user group

A collection of users defined as a single unit and, as a unit, granted or restricted access to DesignSync commands or objects with the access control system. User groups are defined in **Access Administrator**.

user profile

Attributes defined in ProjectSync by an administrator to identify a user. The administrator enters the username, password, and contact information for each team member. ProjectSync can be configured to allow access only to registered users.

V

vault

The repository of the versions, including branched versions, checked in for a particular design object. Note that a vault stores versions of a single design object. A "project vault" refers to the top-level folder of a project and thus contains a vault for each revision-controlled object in the folder. See also DesignSync Data Manager User's Guide: Vaults, Versions, and Branches.

Each DesignSync object has its own vault repository to maintain the branches. DesignSync modules functionality creates an abstraction layer rather than providing direct access to the vault objects. This allows the module to function as an atomic object, rather individual objects.

DesignSync file-based functionality directly manipulates vault objects. This requires each object to be processed individually and enumerated explicitly in such operations as populate, checkout, tag, and checkin.

version

A fixed snapshot of a design object, such as a file or collection, stored in a vault with a numeric identifier. Version numbers are composed of an even number of numeric fields, separated by periods. For example the following are valid version numbers: 1.3, 1.3.1.4, and 1.2.5.1.1.1.See also DesignSync Data Manager User's Guide: Vaults, Versions, and Branches.

version-extended naming

When invoking Advanced Diff from DesignSync, you can specify the files you want to compare using version-extended filenames, which consist of the filename, followed by a semicolon (;), followed by a version number or tag. For example, alu.v;1.2 is version 1.2 of alu.v, and alu.v;golden is the version that is tagged 'golden'. You can also use the following reserved tags:

- **Orig** The version in the vault from which your local version originated; for example, alu.v;Orig.
- **Latest** The most recent version in the vault; for example, alu.v;Latest. Often Latest and Orig are the same version. For example, you fetch alu.v;1.4, which is the most recent version in the vault. Version 1.4 is both Latest and Orig. If your teammate then checks in version 1.5, Orig is still version 1.4, but Latest is now version 1.5.

DesignSync interprets these version-extended filenames and fetches files from the vault as needed, placing them in your DesignSync cache. These cached files may be used by the defined graphical Diff utility ; DesignSync does not use these files and users will not get links to these files if they populate from the cache.

version tag

A tag can be applied to either an object version or a branch. A tag applied to an object version is called a version tag. See DesignSync Data Manager User's Guide: Tagging Versions and Branches.

view

A way to categorize cells. Conceptually, a cell may have many different views--a place and route view, an abstract view, and so on.

In the Milkyway file system, the view is a directory beneath the library directory. The view directory contains all of the cells of that view. (Cells are made up of files in the view directory.) The file system hierarchy has this structure:

Library directory => View directory => Cell view files

W

workspace

A DesignSync workspace is a folder on your local system allows a user to gather various objects, generally under revision control, and work with them as a cohesive unit. Revision Controlled objects can be checked out or populated into the workspace, edited, and then checked in to create the next version on the server vault.

workspace root directory

See workspace root path.

workspace root path

The top folder of your workspace, where information about all modules with base directories at or below that point is stored. In DSDFII, you can manually enter a workspace root path for any modules containing Cadence libraries outside the workspace root directories already known to Cadence. See also module base directory, DesignSync Data Manager DFII User's Guide: Setting the Workspace Root Path.

workspace wizard

A series of dialog boxes, launched by **Revision Control =>Workspace Wizard**, that guide you through the process of creating your own project or joining an existing project. See also Workspace Wizard Overview.

X

Y

Z

Index

A

annotating schematics........................ 72

C

compare 41

configuring....................................... 101

  menus.. 101

conventions.. 3

customer support 119

customizations 111

  compare view object handler.......... 44

  project.. 111

  site 111

D

dssAddFileP function............................ 7

dssBranchCellP function 7

dssBranchCellViewP function 8

dssBranchLibraryP function 9

dssCancelCellViewP function............... 9

dssCancelFileP function..................... 11

dssChangeDefaultsContextP function
  111

dssChangeUserLevelP function 112

dssCheckinCategoryP function 12

dssCheckinCellP function................... 14

dssCheckinCellViewP function........... 16

dssCheckinFileP function................... 18

dssCheckinHierarchyP function 21

dssCheckinLibraryP function.............. 25

dssCheckoutCategoryP function........ 27

dssCheckoutCellP function 29

dssCheckoutCellViewP function......... 30

dssCheckoutFileP function................. 32

dssCheckoutHierarchyP function 35

dssCheckoutLibraryP function............ 39

dssCompareViewsHandlerP function. 44

dssCompareViewsListHandlersP 47

dssCompareViewsP function 41

dssCompareViewsRemoveHandlerP
  function.. 48

dssConfigureLibraryP function 48

dssCreateCellViewP function............. 49

dssDeleteCategoryP function............. 50

dssDeleteCellP function 52

dssDeleteCellViewP function 54

dssDeleteFileP function...................... 56

dssDeleteLibraryP function 59

dssDeleteTemporaryViewsP function. 60

dssDeleteVersionP function 61

dssEnableDebugP function 112

dssExecuteTclP function 113

dssFetchCellViewVersionP function... 61

dssFetchLockedP function 62

dssGetFileTagsP function 65

dssGetFileVersionP function 66

dssGetFileVersionsP function 68

dssGetTagListP function 69

dssGetViewPathP function................. 71

dssGetViewTagsP function 71

dssGetViewVersionP function 72

dssHelpP function 114

dssInit.il file....................................... 101

dssIsFileLockedP function.................. 75

dssIsViewLockedP function................ 76

dssJoinLibraryP function 77

dssLibraryStatusP function................. 79

dssListHierarchyP function................. 81

dssMenuAddItemP function 101

dssMenuAddValidItemP function 102

dssMenuListItemsP function 104

dssMenuListMenuP function 104

dssMenuLoadConfigP function 105

dssMenuRefreshP function 105

dssMenuRemoveItemAllP function .. 106

dssMenuRemoveItemP function 106

dssMenuRemoveValidItemP function
.... 107

dssMenuSaveConfigP function 108

dssRefreshWindowBannerP function108

dssTagCategoryP function 83

dssTagCellP function 85

dssTagCellViewP function.................. 86

dssTagFileP function.......................... 87

dssTagHierarchyP function 89

dssTagLibraryP function..................... 93

dssUnlockCellViewP function............. 94

dssUnlockFileP function 95

dssViewDataSheetP function 96

dssViewVersionHistoryP function....... 98

E

error handling 5

F

functions...................................... 3, 111

dssAddFileP 7
dssBranchCellP 7
dssBranchCellViewP 8
dssBranchLibraryP 9
dssCancelCellViewP........................ 9
dssCancelFileP.............................. 11
dssChangeDefaultsContextP........ 111
dssChangeUserLevelP 112
dssCheckinCategoryP 12
dssCheckinCellP............................ 14
dssCheckinCellViewP.................... 16
dssCheckinFileP 18
dssCheckinHierarchyP 21
dssCheckinLibraryP....................... 25
dssCheckoutCategoryP 27
dssCheckoutCellP 29
dssCheckoutCellViewP.................. 30
dssCheckoutFileP.......................... 32
dssCheckoutHierarchyP 35
dssCheckoutLibraryP..................... 39
dssCompareViewsHandlerP........... 44
dssCompareViewsListHandlersP.... 47
dssCompareViewsP 41
dssCompareViewsRemoveHandlerP 48
dssConfigureLibraryP 48
dssCreateCellViewP....................... 49
dssDeleteCategoryP....................... 50
dssDeleteCellP 52
dssDeleteCellViewP 54
dssDeleteFileP 56
dssDeleteLibraryP 59
dssDeleteTemporaryViewsP 60
dssDeleteVersionP 61
dssEnableDebugP........................ 112
dssExecuteTclP 113
dssFetchCellViewVersionP............. 61
dssFetchLockedP 62
dssGetFileTagsP 65
dssGetFileVersionP 66
dssGetFileVersionsP 68
dssGetTagListP 69
dssGetViewPathP........................... 71
dssGetViewTagsP 71
dssGetViewVersionP 72
dssGetViewVersionsP 74
dssHelpP 114

dssIsFileLockedP............................ 75

dssIsViewLockedP.......................... 76

dssJoinLibraryP 77

dssLibraryStatusP........................... 79

dssListHierarchyP........................... 81

dssMenuAddItemP 101

dssMenuAddValidItemP 102

dssMenuListItemsP 104

dssMenuListMenuP 104

dssMenuLoadConfigP 105

dssMenuRefreshP 105

dssMenuRemoveItemAllP 106

dssMenuRemoveItemP 106

dssMenuRemoveValidItemP 107

dssMenuSaveConfigP 108

dssRefreshWindowBannerP......... 108

dssTagCategoryP........................... 83

dssTagCellP 85

dssTagCellViewP............................ 86

dssTagFileP.................................... 87

dssTagHierarchyP 89

dssTagLibraryP............................... 93

dssUnlockCellViewP....................... 94

dssUnlockFileP............................... 95

dssViewDataSheetP....................... 96

dssViewVersionHistoryP................. 98

G

getting help...................................... 117

H

help 117

L

loading SKILL functions.................... 101

M

menus 101

customizing.................................. 101

P

PDF file 118

of API reference............................ 118

printing documentation..................... 118

production documentation................ 118

accessing...................................... 118

S

schematics 72

adding version number 72

SKILL functions 101

loading... 101

synctrace command 5

syntax conventions............................... 3

T

tracing 5

V

version number 72, 74

  adding to schematics...................... 72

views

  compare.. 41