ENOVIA DesignSync
DFII SKILL Programming Interface Guide

3DEXPERIENCE 2022

Table Of Contents

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.

Note on using this guide: References from the *ENOVIA Synchronicity DesignSync Data Manager DFII SKILL Programming Interface Guide* to the *ENOVIA Synchronicity Command Reference* guide always link to the ALL version of the guide, which contain information about all working methodologies for DesignSync. For more information about the available working methodologies, see ENOVIA Synchronicity Command Reference.

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. If a null string is specified, the command will use smart module detection to determine the target module. For more information on referring to modules or on how DesignSync uses smart module detection to determine the target module, see *ENOVIA Synchronicity DesignSync Data Manager User's Guide:* Specifying Module Objects for Operations and Understanding Smart Module Detection.
`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. By using a SKILL list containing the path to a file, you can specify a comment file. Comment files support UTF-8 compliant multibyte characters. If DesignSync DFII is configured to require a comment of a particular length, it should be noted that each byte in a multibyte character counts individually towards the comment length .
`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`.

Example

This example shows a checkin that uses a comment file.

```
dssCheckinCategoryP "df2test" "inv2" "layout" ?force t ?comment
(list "/my/comments")
```

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. By using a SKILL list containing the path to a file, you can specify a comment file. Comment files support UTF-8 compliant multibyte characters. If DesignSync DFII is configured to require a comment of a particular length, it should be noted that each byte in a multibyte character counts individually towards the comment length .
`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`.

Example

This example shows a checkin that uses a comment file.

```
dssCheckinCellP"df2test" "inv2" "layout" ?force t ?comment (list
"/my/comments")
```

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. By using a SKILL list containing the path to a file, you can specify a comment file. Comment files support UTF-8 compliant multibyte characters. If DesignSync DFII is configured to require a comment of a particular length, it should be noted that each byte in a multibyte character counts

	individually towards the comment length .
`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`.

Example

This example shows a checkin that uses a comment file.

```
dssCheckinCellViewP "df2test" "inv2" "layout" ?force t ?comment
(list "/my/comments")
```

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. By using a SKILL list containing the path to a file, you can specify a comment file. Comment files support UTF-8 compliant multibyte characters. If DesignSync DFII is configured to require a comment of a particular length, it should be noted that each byte in a multibyte character

	counts individually towards the comment length .
`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`	Specifies 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. If you have enabled checkin of new objects and not specified a module context, the commandt uses smart module detection to determine the target module. For more information on how smart module detection determines the target module, see *ENOVIA Synchronicity DesignSync Data Manager User's Guide*: Understanding Smart Module Detection. **Note:** You can only specify one module context value..

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`.

Example

This example shows a checkin that uses a comment file.

```
dssCheckinFileP "/CHDX2/Rev1/chxr1" ?force t ?comment (list
"/my/comments")
```

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]
  [?includeConfigs g_includeConfigs]
  [?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 t`l_viewNames` argument. Then, DesignSync DFII descends into the views as appropriate.

For design views, DesignSync DFII scans the hierarchy, beginning with the top-level cell views you specify using the `l_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.

For config views, DesignSync traverses the hierarchy specified by the view. Because it uses the information contained in the config view, it does not use the arguments `l_switchList`, `t_switchUsing` and `l_stopList`. You can limit which views or libraries are tagged by using the `g_processViews S_switchLibChoice` arguments.

Notes:

- For DesignSync DFII to scan the hierarchy, the cells must be in your local workspace.
- DesignSync provides support for operating both on design views and config views, but you cannot specify both types of views within the same operation.
- 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.

Note: The **Switch Using, Switch List, and Stop List** fields are not applicable to "config" views.

DesignSync provides support for operating both on design views and config views, but you cannot specify both types of views within the same operation.

`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.

This field is not applicable when specifying a config view.

`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.

This field is not applicable when specifying a config view.

`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.

This field is not applicable when specifying a config view.

`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)
- `switchList`: For config views, use the switch view and the switch list defined within the config view. If there are sub-configs, then the switch list of the sub-configs is used within those sub-configs. For non-config views, use the value specified for the `l_switchList` option.
- List of views to process.

`g_includeConfigs` Specifies whether the config view cells are included in the operation.

- `nil`: Only the design cells are included in the operation. The hierarchy definition cells for the config view are omitted.
- `t`: Operate on the design cells and the hierarchy definition (config) cells. (Default)

This argument is silently ignored if the specified view is not a config view.

`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.

By using a SKILL list containing the path to a file, you can specify a comment file. Comment files support UTF-8 compliant multibyte characters. If DesignSync DFII is configured to require a comment of a particular length, it should be noted that each byte in a multibyte character counts individually towards the comment length .

`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`.

Example

This example shows a checkin that uses a comment file.

```
dssCheckinHierarchyP "df2test" "inv2" "layout" ?force t ?comment
(list "/my/comments")
```

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. By using a SKILL list containing the path to a file, you can specify a comment file. Comment files support UTF-8 compliant multibyte characters. If DesignSync DFII is configured to require a comment of a particular length, it should be noted that each byte in a multibyte character counts individually towards the comment length .
`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 specifying a module, you must use two separate checkins to check objects to two different modules in the same workspace. If you do not specify a module context, the operation applies to all objects specified. If you have enabled checkin of new objects and not specified a module context, the command uses smart module detection to determine the target module. For more information on how smart module detection determines the target module, see *ENOVIA Synchronicity DesignSync Data Manager User's Guide*: Understanding Smart Module Detection.
`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`.

Example

This example shows a checkin that uses a comment file.

```
dssCheckinLibraryP "df2test" ?force t ?comment (list
"/my/comments")
```

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_vaultVersion` option to specify a module member version. `t_vaultVersion` and `t_tag` are mutually incompatible.
`t_vaultVersion`	Version number of the module member object to be checked out. **Note:** The `t_vaultVersion` 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]
  [?includeConfigs g_includeConfigs]
  [?processFiles gS_processFiles] [?silent g_silent]
  [?background g_background]
)
=> nil/(x_pass x_fail)
```

Description

Checks out objects of a design hierarchy.

For design views, DesignSync DFII scans the hierarchy, beginning with the top-level cell views you specify using the `l_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.

For config views, DesignSync traverses the hierarchy specified by the view. Because it uses the information contained in the config view, it does not use the arguments `l_switchList`, `t_switchUsing` and `l_stopList`. You can limit which views or libraries are checked out by using the `g_processViews S_switchLibChoice` arguments.

Notes:

- For DesignSync DFII to scan the hierarchy, the cells must be in your local workspace.
- DesignSync provides support for operating both on design views and config views, but you cannot specify both types of views within the same operation.
- 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. **Note:** The **Switch Using, Switch List, and Stop List** fields are not applicable to "config" views. DesignSync provides support for operating both on design views and config views, but you cannot specify both types of views within the same operation.
`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. This field is not applicable when specifying a config view.
`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. This field is not applicable when specifying a config view.
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.

This field is not applicable when specifying a config view.

`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)
- `switchList`: For config views, use the switch view and the switch list defined within the config view. If there are sub-configs, then the switch list of the sub-configs is used within those sub-

configs. For non-config views, use the value specified for the `l_switchList` option.
- List of views to process.

`g_includeConfigs` Specifies whether the config view cells are included in the operation.

- `nil`: Only the design cells are included in the operation. The hierarchy definition cells for the config view are omitted.
- `t`: Operate on the design cells and the hierarchy definition (config) cells. (Default)

This argument is silently ignored if the specified view is not a config view.

`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
?mergeAddedProc s_mergeAddedProc ?mergeDeletedProc
s_mergeDeletedProc ?diffProc ls_diff ?extractProc s_extract
?figsProc s_figsProc)

=> t
```

Description

The `dssCompareViewsHandlerP` 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. **Note:**If you want the differences found by a custom compare procedure to be merged, the compare procedure must also return the merge procedure call. The call returned must include both the procedure name AND the arguments to provide to the procedure. For more information on using the calls, see the examples provided with DSDFII.
s_mergeAddedProc	The name of the procedure called to merge an object that is present in the first cellview into the second cellview. The supplied function takes two arguments: the object from the first cell view, and the second cell view into which the object is added or changed.
s_mergeDeletedPro	The name of the procedure called to merge (delete) an object that is present in the second cellview but not in the first. The supplied function takes one argument: the object present only in the second cellview.
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 highlighted. 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 identify 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]
  [?includeConfigs g_includeConfigs]
  [?processFiles gS_processFiles] [?silent g_silent]
```

```
)
=> l_result
```

Description

Lists the objects of a design hierarchy in the workspace.

For design views, DesignSync DFII scans the hierarchy, beginning with the top-level cell views you specify using the `l_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.

For config views, DesignSync traverses the hierarchy specified by the view. Because it uses the information contained in the config view, it does not use the arguments `l_switchList`, `t_switchUsing` and `l_stopList`. You can limit which views or libraries are listed by using the `g_processViews S_switchLibChoice` arguments.

Notes:

- For DesignSync DFII to scan the hierarchy, the cells must be in your local workspace.
- DesignSync provides support for operating both on design views and config views, but you cannot specify both types of views within the same operation.
- 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.

	Note: The **Switch Using, Switch List, and Stop List** fields are not applicable to "config" views. DesignSync provides support for operating both on design views and config views, but you cannot specify both types of views within the same operation.
`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`	This field is not applicable when specifying a config view. 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`	This field is not applicable when specifying a config view. 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`	This field is not applicable when specifying a config view. 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)
- `switchList`: For config views, use the switch view and the switch list defined within the config view. If there are sub-configs, then the switch list of the sub-configs is used within those sub-configs. For non-config views, use the value specified for the `l_switchList` option.
- List of views to process.

`g_includeConfigs` Specifies whether the config view cells are included in the operation.

- `nil`: Only the design cells are included in the operation. The hierarchy definition cells for the config view are omitted.
- `t`: Operate on the design cells and the hierarchy definition (config) cells. (Default)

This argument is silently ignored if the specified view is not a config view.

`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.

dssSetModuleSelector

```
dssSetModuleSelectorP(

  t_moduleName t_selector [?silent g_silent]

)

=> t/nil
```

Description

Sets the persistent selector for the module. This allows the user to work in a module member tag environment and use blended selectors. For more information on blended selectors and module members tags, see the *DesignSync User's Guide*: Module Member Tags.

Arguments

`t_moduleName` — The workspace address of the module. This can be a simple module name, module instance name or full workspace address. If a module name or module instance name is given, it must be unique within the set of modules that are known within the DSDFII session.
For more information on referring to modules or on how DesignSync uses smart module detection to determine the target module, see *ENOVIA Synchronicity DesignSync Data Manager User's Guide:* Specifying Module Objects for Operations and Understanding Smart Module Detection.

`t_selector` — Selector specifying the new persistent selector for the module.

	This selector format is validated before being applied.
g_silent	Run silently (t). (Default) Command reports output (nil).

Value Returned

Returns t if the persistent selector is modified. Returns nil if there is an error resulting in no change to the persistent selector.

dssRollbackCellViewP

```
dssRollbackCellViewP(
t_libName t_cellName t_viewName
[?tag g_tag | ?vaultVersion g_vaultVersion]
[?silent g_silent] [?comment t_comment]
)

=> (1 0)/nil
```

Description

Creates a rollback version of a cell view; taking a previously checked in view and making it the latest view on the branch indicated by the workspace selector.

This command follows the methodology defined in the options for the client; either the skip method or the lock method. For more information on rollback, see the *DSDFII User's Guid*e: Rollback Cell Views.

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_vaultVersion option to specify a module member version. t_vaultVersion and t_tag are mutually incompatible.
t_vaultVersion	Version number of the module member object to be checked

	out. **Note:** The `t_vaultVersion` and `t_tag` options are mutually incompatible.
`g_silent`	Run silently (`t`). (Default)
`t_comment`	Check in comment for the rollback version. This comment should explain the reason for the rollback. The comment is pre-pended automatically with the rollback information in the following form: `Rollback checkin from <`*`library`*`> <`*`cell`*`> <`*`view`*`> version <`*`version`*`> [RETURN]` *<user entered comment>*

Value Returned

The `dssRollbackCellViewP` function lets you rollback a single cell view only, so the returned list is (1 0) if the rollback is successful. The function raises an error if argument checking fails. In all other failure cases, the function either raises an error or returns `nil`.

dssSwapReplaceP

```
dssSwapReplaceP(
t_moduleName t_selector [?force g_force] [?silent g_silent]
[?mcachemode t_mode]
)

=> t/nil
```

Description

Replaces the version of a module in the workspace with a different version of the same module. For more information on understanding this functionality, see the *DesignSync Data Manager User's Guide*: Edit-in-Place Methodology.

Arguments

`t_moduleName`	Module name. (Required)
`t_selector`	Selector specifying the new persistent selector for the module. This selector format is validated before being applied. (Required)
`g_force`	Overwrite the file even if it is locally modified (`t`).
`g_silent`	Run silently (`t`). (Default)
`t_mode`	Specifies the mcache mode. The mcache mode value is not

checked at the DSDFII level, but rather by the underlying DesignSync command. The mcache mode must be one of the valid values, "`link`," "`copy`," "`server`."

Value Returned

Returns t if the module replace is successful. Returns nil if there is an error resulting in no change to the module.

dssSwapRestoreP

```
dssSwapRestoreP(
t_moduleName t_parent [?force g_force] [?silent g_silent]
[?mcachemode t_mode]
)

=> t/nil
```

Description

Restores to selected sub-module to the selector by which this sub-module is referenced from its parent module or modules. For more information on understanding this functionality, see the *DesignSync Data Manager User's Guide*: Edit-in-Place Methodology.

Arguments

`t_moduleName`	Module name. (Required)
`t_parent`	Name of the parent module, usually in the form of the module instance name. Value can also be supplied as a null ("") string. (Default)
`g_force`	Overwrite the file even if it is locally modified (`t`).
`g_silent`	Run silently (`t`). (Default)
`t_mode`	Specifies the mcache mode. The mcache mode value is not checked at the DSDFII level, but rather by the underlying DesignSync command. the mcache mode must be one of the valid values, "`link`," "`copy`," "`server`."

Value Returned

Returns t if the module restore is successful. Returns nil if there is an error resulting in no change to the module hierarchy.

dssSwapShowP

```
Type todssSwapShowP()
```

```
=> t
```

Description

Displays a list of all the modules in a workspace that have had their versions replaced with different module versions. From the display window, you can perform operations, such as restoring a module to version corresponding to the persistent selector on the workspace. For more information on understanding this functionality, see the *DesignSync Data Manager User's Guide*: Edit-in-Place Methodology.

Value Returned

Displays the result window with the output from the command. The value of this command cannot be used in post processing.

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] [?recursive g_recursive] [?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_recursive	For folders, tag the contents of the folder recursively. If performed on a module folder, the folders are traversed recursively, but the module hierarchy is not. For module instances, tag the contents of the module hierarchy recursively.
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]
  [?includeConfigs g_includeConfigs]
  [?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. DesignSync provides two mechanisms to identify the cells in a design hierarchy.

For design views, 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.

For config views, DesignSync traverses the hierarchy specified by the view. Because it uses the information contained in the config view, it does not use the arguments `l_switchList`, `t_switchUsing` and `l_stopList`. You can limit which views or libraries are tagged by using the `g_processViews S_switchLibChoice` arguments.

Notes:

- For DesignSync DFII to scan the hierarchy, the cells must be in your local workspace.
- DesignSync provides support for operating both on design views and config views, but you cannot specify both types of views within the same operation.
- 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. **Note:** The **Switch Using, Switch List, and Stop List** fields are not applicable to "config" views. DesignSync provides support for operating both on design views and config views, but you cannot specify both types of views within the same operation.
`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.
	This field is not applicable when specifying a config view.
`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.
	This field is not applicable when specifying a config view.
`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.

This field is not applicable when specifying a config view.

`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)
- `switchList`: For config views, use the switch view and the switch list defined within the config view. If there are sub-configs, then the switch list of the sub-configs is used within those sub-configs. For non-config views, use the value specified for the `l_switchList` option.
- List of views to process.

`g_includeConfigs`
Specifies whether the config view cells are included in the operation.

- `nil`: Only the design cells are included in the operation. The hierarchy definition cells for the config view are omitted.
- `t`: Operate on the design cells and the hierarchy definition (config) cells. (Default)

This argument is silently ignored if the specified view is not a config view.

`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
```

dssViewWhereUsedP

```
dssViewWhereUsedP(
 t_fileName | t_libraryName t_cellName t_viewName
    [?version t_version] [?showtags t_showtags]
)
=> t/nil
```

Description

Displays the whereused information for where the specified version of the module member is used..

Arguments

`t_fileName` File name. (Required unless you specify a cell view:

	`t_libName t_cellName t_viewName`.) 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`.
`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`.)
`t_version`	Version specifier. The legal values are: **all** - show whereused results for all member versions. **current** - show whereused results for the current member version populated in the workspace. **<*Selector_List*>** - a comma separated list of selected versions. This list is not validated prior to execution.
t_showtags	Specifies whether tag information is displayed and optionally restricts the output to immutable tagged version or tagged versions. The legal values are: **all** - Displays all module versions containing the member version and all tags associated with the module or member version. (Default) **none** - Displays all module versions containing the member version, but does not display tag information.. **version** - Displays any reference location that has a version tag and the name of the tag. **immutable** - Displays only module versions containing the member version that are tagged with an immutable tag and the name of the immutable tag. **Note:** Using the -showtags immutable option may not display all versions in which an immutable tag is used. The whereused command queries for all the whereused information but filters the display from the starting point until it reaches the last immutable tag in a reference tree. **member** - Displays any module version containing the member version that has a member tag and the name of the tag.

Value Returned

Displays the result window with the output from the command. The value of this command cannot be used in post processing.

Menu Customization Functions

Customizing the Menu

You (or your project leader) can configure the menus that are part of the interfaces provided with the DSDFII integration, selectively removing, adding, or reordering submenus and commands. You can customize menus and toolbars on the main Command Interface Window, the Status Browser windows, and the editor windows, including the Hierarchy Editor. For example, if your team uses scripts to place libraries under revision control, your project leader might remove **Configure Library** from the Synchronicity menu on the Command Interface Window. 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,menu location (CIW, Status Browser, Hierarchy editor, etc.), 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 menues. These functions are defined when `dssInit.il` is loaded (see the DesignSync Data Manager DFII User's Guide: Loading the DesignSync Integration into DFII).

Related Topics

Customizing Menu Items

Customizing Menu Items

Each DSDFII interface, the Command Interface Window, the Design Editor, the Hierarchy Editor and the Status Browser, provide its own set of menus that can be customized. Using the functions provided in this section, you can add, remove or change which menu items are displayed on any of the interface menus.

Available Menus

Menu Name	Description
CIWNovice	Command Interface Window menu displayed in Novice mode.
CIWAdvanced	Command Interface Window menu displayed in Advanced mode.
CIWExpert	Command Interface Window menu displayed in Expert mode.
DENoviceFull	Design Editor full menu displayed in Novice mode.
DEAdvancedFull	Design Editor full menu displayed in Advanced mode.

DEExpertFull	Design Editor full menu displayed in Expert mode.
DENoviceShort	Design Editor short menu displayed in Novice mode.
DEAdvancedShort	Design Editor short menu displayed in Advanced mode.
DEExpertShort	Design Editor short menu displayed in Expert mode.
SBBannerNovice	Status Browser "banner," the menus at the top of the Status Browser, in Novice mode.
SBBannerAdvanced	Status Browser "banner," the menus at the top of the Status Browser, in Advanced mode.
SBBannerExpert	Status Browser "banner," the menus at the top of the Status Browser, in Expert mode.
SBContextNovice	Status Browser Context Menus displayed when the Status Browser is open in Tree View in Novice mode.
SBContextAdvanced	Status Browser Context Menus displayed when the Status Browser is open in Tree View in Advanced mode.
SBContextExpert	Status Browser Context Menus displayed when the Status Browser is open in Tree View in Expert mode.
SBLibraryNovice	Status Browser Context Menu for libraries selected in List View in Novice mode.
SBLibraryAdvanced	Status Browser Context Menu for libraries selected in List View in Advanced mode.
SBLibraryExpert	Status Browser Context Menu for libraries selected in List View in Expert mode.
SBCategoryNovice	Status Browser Context Menu for categories selected in List View in Novice mode.
SBCategoryAdvanced	Status Browser Context Menu for categories selected in List View in Advanced mode.
SBCategoryExpert	Status Browser Context Menu for categories selected in List View in Expert mode.
SBCellNovice	Status Browser Context Menu for cells selected in List View in Novice mode.
SBCellAdvanced	Status Browser Context Menu for cells selected in List View in Advanced mode.
SBCellExpert	Status Browser Context Menu for cells selected in List View in Expert mode.
SBViewNovice	Status Browser Context Menu for views selected in List View in Novice mode.
SBViewAdvanced	Status Browser Context Menu for views selected in List View in Advanced mode.
SBViewExpert	Status Browser Context Menu for views selected in List View in Expert mode.

SBFileNovice	Status Browser Context Menu for files selected in List View in Novice mode.
SBFileAdvanced	Status Browser Context Menu for files selected in List View in Advanced mode.
SBFileExpert	Status Browser Context Menu for files selected in List View in Expert mode.
SBModuleNovice	Status Browser Context Menu for modules selected in List View in Novice mode.
SBModuleAdvanced	Status Browser Context Menu for modules selected in List View in Advanced mode.
SBModuleExpert	Status Browser Context Menu for modules selected in List View in Expert mode.
HEDNovice	Hierarchy editor menu displayed in Novice mode.
HEDAdvanced	Hierarchy editor menu displayed in Advanced mode.
HEDExpert	Hierarchy editor menu displayed in Expert mode.

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. For a list of available menu names see Customizing Menu Items.(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.

Arguments

`t_menu`	The name of a menu, To view the structure of all menus, specify `all`. (Default) For a list of available menu names see Customizing Menu Items.
`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. (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/")
```

dssStatusBrowserStatusIconU

```
dssStatusBrowerStatusIconU(
  r_item l_props s_icon
)
=> t_result
```

Description

Extends the Status Browser icon system to include an icon for an element in the status browser that does not have an icon by default or to select a different icon for a defined element. This is an optional function. If the function is not defined, then the default icons are used. If the function is defined, then it returns an identifier to indicate the icon to be used for an individual item.

For example,if instead of using the provided red and green icons to indicate whether an item is locked or unlocked, you could use this function to specify icons that are different colors or use a padlock, similar to the one used by the DesSync GUI.

Note: If this function is defined, it will be called frequently and on each object shown. Therefore, it must be implemented to be efficient and fast, to avoid performance issues.

Arguments

r_item	The "treeItemStruct" item to which this icon applies.
`l_props`	The property list containing the values for each column for this item. The keys in this property list are the report values returned by the DesignSync ls command. For information on data keys, see the Report Data Keys Table in the ls command in the *Synchronicity Command Reference.*
s_icon	A symbol matching the name of an entry in dssIconsMap that would be used for this object by default.

Value Returned

`t_result`	One of the following values must be returned • A SKILL symbol that identifies one of the standard DSDFII icons, from the dssIconsMap. • A Cadence icon identifier/size list, similar, for example, to the value returned by the hiLoadIconFile function. • A null value indicating that no icon is shown for the object.

Related Topics

DesignSync Data Manager DFII User's Guide: Customizing the Status Browser Icons

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.

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 through 3ds support site. 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

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

Index