ENOVIA DesignSync Command Reference All Volume 2

3DEXPERIENCE 2022

Introduction to the PDF version of the DesignSync Command Reference

Because of the quantity of information included in the DesignSync Command Reference, the PDF version of the book has been divided into two volumes.

Volume 1 contains the following information:

- Fundamental Topics
- Client Applications
- Client Shell Control
- Module-Based Design
- Legacy Module-Based Design

Volume 2 contains the following information:

- File-Based Design
- Enterprise DesignDevelopment
- URL Sync Object Model
- TCL Interface
- Third-Party Integrations
- Administration
- ProjectSync Data Manipulation
- Glossary

Table of Contents

ENOVIA Synchronicity Command Reference

This document contains command descriptions for all ENOVIA Synchronicity DesignSync and ProjectSync® commands. You can run most DesignSync commands from any DesignSync client.

The commands in this reference, along with the Tcl scripting language, are referred to as Synchronicity tcl (stcl). You can include these commands in stcl scripts. For DesignSync, you can create scripts for clients and servers (SyncServers). For ProjectSync, you create server scripts. See the Synchronicity stcl Programmer's Guide to learn how to use the Synchronicity commands in Tcl scripts.

Using this Guide with Different Methodologies

DesignSync features three command references. This command reference is for a mixed environment containing any combination of: modules-based, legacy modules-based, or files-based objects.

Module Based Commands

The module-based command set supports working in the modern modules methodology. Modules are processed as a single versionable DesignSync vault object that contains individual module members which are also versionable DesignSync objects. Modules can also contain references to other versionable DesignSync objects. In this guide, specific module-based information is called out in individual sections. Any sections that do not specify otherwise, are applicable to modules. If you are only working in a modules-based environment, and you do not wish to see any non-modules information, use the ENOVIA Synchronicity Command Reference: Modules Only.

Within the Module-Based Design section, the commands only include links relevant to the modules environment. The topics, however, contain all the information for all methodologies.

Legacy Module Based Commands

The legacy module based command set supports working in the legacy modules methodology. Legacy modules provided some of the functionality now available in modern modules built on top of the files-based methodology. For this reason, the majority of the files-based information is also applicable to legacy modules. When using the guide, if there is no specific legacy-module based information called out, the legacy modules follows the files-based methodology. There is no separate guide containing the

legacy-modules based command information. All the information for legacy modules is contained in this guide.

File Based Commands

The file based command set supports working with individually managed file objects which each have a vault object on DesignSync server that can be directly manipulated and referenced. In this guide, specific file based commands and sections are called out. Any sections that do not specify otherwise are applicable to files based objects. If you are only working in a file-based environment, and you do not wish to see any non-file-based information, use the ENOVIA Synchronicity Command Reference: File-Based Only.

Within the File-Based Design section, the commands only include links relevant to the file-based environment. The topics, however, contain all the information for all methodologies.

Organization of the Command Reference

The Synchronicity commands are ordered by methodology and function within the Command Reference.

The command reference sections are organized into the following sections:

- Fundamental Topics - contains the commands that discuss the core concepts underlying the DesignSync system. Understanding these concepts provides a foundation that allows you to properly construct DesignSync Commands and maximize their functionality. These topics include Overview of HCM (Legacy Module) Commands, Overview of Module Commands, Understanding Fetch Preferences, Using Interrupt, Using Revision Control Keywords, selectors, server-side, wildcards.
- Client Applications - contains the commands that launch DesignSync clients. These topics include DesSync (graphical user interface, or GUI), dss (DesignSync shell), dssc (concurrent version of dss), stcl (Synchronicity Tcl shell), and stclc (concurrent version of stcl). See DesignSync Help: DesignSync Command Line Shells for details about the shells, as well as the types of command line editing supported by each shell.
- Client Shell Control - contains the commands used within the DesignSync clients to control the clients.
- Methodology Commands - consisting of:
    - Workspace Setup - commands used to initially set up a workspace.
    - Primary Revision Control - primary commands used daily by the user to manage their data.

    - Advanced Revision Control - advanced commands used less often by the user to support advanced revision control functionality.
    - Navigational - commands that allow you to move around within the workspace or the server to locate your files.
    - Informational - commands that provide information about the revision controlled objects, or the contents of the files.

  Note: The Modules-Based Design methodology structure also includes

    - Module Hierarchy Management - commands that provide information about building and maintaining your module hierarchy.
    - Workflows - commands that provide information about work-flows built on top of modules to provide a customized working methodology, like SiTaR.

  Note: The Legacy Module-Based Design workflow commands are not organized into sub-categories.

- Enterprise Design Development - commands that support enterprise development; the creation and maintenance of development areas, enterprise object management, and mcache settings for shared developments.
- URL Sync Object Module - contains the commands that allow you to access (view and modify) the Synchronicity Object Model information.
- TCL Interface - contains the commands that provide additional TCL scripting functionality.
- Third-Party Integrations - contains the commands that provide an interface into theDSDFII integration.
- Administration - contains the commands that provide administration resources, such as data replication, caching, mirrors, authentication, command defaults setup, triggers, etc.
- ProjectSync Data Manipulation - contains the commands that provide an interface into the DesignSync Web interface, including note and notetype manipulation, property type manipulation, etc.

Syntax Description

Every command description has a SYNOPSIS section that shows the syntax for the command. Material within square brackets [ ] is optional. Material within curly brackets { } is required. A vertical bar | indicates mutual exclusion. For example, `[ -keep | -lock ]` means that you can use the `-keep` option or the `-lock` option, but not both.

The command options descriptions are in alphabetical order in the TOC and within the options section. In the syntax line, however, they are in approximate alphabetical order, to allow exclusive options to remain together for readability.

Accessing Command Descriptions from Client Shells

The command descriptions in the **Synchronicity Command Reference** are identical to the command-line help you can access from any DesignSync client using the `help` command or `-help` option. For example, you can enter either of the following commands to get help for the `co` command:

```
dss> help co

dss> co -help
```

You can also launch this file from any DesignSync client by using the `webhelp` command. For example,

```
dss> webhelp co
```

Opens your default web browser on the co command page.

Note: Both the help and webhelp commands respect the DesignSync methodology specified in SyncAdmin. When a methodology is specified, Designsync shows the custom help for the module specified. For more information on specifying methodology, see the *ENOVIA Synchronicity DesignSync Administrator's Guide*: General Options.

File-Based Design

Workspace Setup

Enterprise Design Development Area

sda cd

sda cd Command

NAME

```
sda cd              - Change development area and launch a tool command
```

DESCRIPTION

```
This command allows the user to launch a tool from a development area they
have created via "sda mk" or joined via "sda join". The tool runs using the
development setting defined for the area.

The sda cd command performs the following sequence of actions:
1. If the -update option is selected, updates the development
   instance directory associated with an external development area.

2. Sets up the environment by setting the following environment
   variables:
     o SYNC_DEVAREA_DIR - set to the requested development area
       directory.
     o SYNC_DEVAREA_TOP - set to the leaf name of the top module or
       directory in the development area.
     o SYNC_DEV_ASSIGNMENT - set to the assignment associated with
       the development area.
     o SYNC_DEVELOPMENT_DIR - set to the top of the development instance
       directory.
     o SYNC_PROJECT_CFGDIR - set to the directory holding the
       development setting for the assignment associated with the
       development area.
     o SYNC_WS_DEVAREA_TOP - set to the leaf name of the top module
       or directory in the development area. This variable can then
       be used for the starting directory in any commands you
       construct within the specified tool.

3. Runs all of the set up scripts defined for the tools associated
   with the development area. Running all the scripts is required to
   support inter-tool dependencies and shell tools.
   Note: When a shell is defined as a tool, it should be defined to
   ignore the startup script for the shell. Any aliases, etc. defined
   in the startup script will not be available; however when a tool
   suite is defined, the admin can specify a script with the desired
   environment settings.
```

4. Sets the current directory for the tool to the starting directory. The starting directory is the directory defined in the tool's definition. If no starting directory is specified, then the directory defined in the tool suite is used. If no starting directory is specified in the tool suite either, the development area is used.
   The starting directories can be specified with environment variables and may be relative to the development area.

5. Starts the requested tool. If the tool is graphical, the tool is spawned (detached) from sda. If the tool is non-graphical, on UNIX, the tool runs in the same shell as sda.

   Note: When a non-graphical tool is started, the sda process ends.

If you run the command without specifying a development area or a tool, or the user specified an ambiguous argument, the command starts in interactive mode. In interactive mode, the user is prompted for the command arguments and options needed. Any arguments specified with the -gui command option are passed to the GUI and the appropriate fields are selected on the "Change Area" tab.

SYNOPSIS

```
sda cd [<area_name>] [<tool>] [-development <name>] [-gui]
       [-suite <suite_name>] [-[no]update] [-version <version>]
```

ARGUMENTS

Development Area Name

area_name	The development area name of the DesignSync Development. This argument is required and the development area must already exist.

Tool

tool	The tool name specified must be a tool that is defined for use with the specified development area. The list of available tools can be viewed from the development instance for the assignment associated with the area. Note: When a shell is defined as a tool, it should

```
be defined to ignore the startup script for the
shell.  Any aliases, etc. defined in the startup
script will not be available.
```

OPTIONS

-development

```
-development      Specify the name of the development if the area name is
 <name>           not unique for the user. Area names are unique
                 within a development for a given user, but are not
                 required to be unique across all developments.
```

-gui

```
gui               Starts the sda graphical user interface mode
                  with the "Change Area" tab selected.

                  If this option is used with the tool argument, the
                  tool argument is silently ignored.
```

-suite

```
-suite <suite>    Specify the suite name for the tool suite, if the
                  tool name is not unique across all tool suites for
                 the development assignment.
```

-[no]update

```
-[no]update       Specifies whether the development instance
                  definition should be updated, if it is an external
                  area.

                  -noupdate does not update the external development
                  instance from the server before setting the
                  environment variables for the area and starting the
                  tool. (Default when the development setting
                  is 'Mirror=False')
```

```
 -update performs the update of the external area
 before performing any other actions. (Default when
 'Mirror=True')

 If the area is not an external area and this option
is specified, the tool exits without launching the
tool.

Note: If -update is explicitly specified, and no
       tool is specified, DesignSync assumes the
       desired action is the update and does not
       prompt for tool in interactive mode.
```

-version

```
-version          Specify the version number of the tool suite if the
 <version>        tool suite name is not unique within the
                  development assignment. This option must be
specified if there are multiple tools with the same
name in multiple tool suites with the same name.
```

RETURN VALUE

```
There is no TCL return value for this command.
```

SEE ALSO

```
sda gui, sda join, sda ls, sda mk, sda rm
```

EXAMPLES

- Running sda cd in Interactive Mode
- Running sda cd in non-interactive mode

Running sda cd in Interactive Mode

```
This example runs sda cd in interative mode, supplying no
arguments. It is run from a Windows client and launches the
DesignSync GUI which is configured as a tool for this development
area.
Note that the list of areas is prefixed with the development name for
ease of idenfitication.

C:\workspaces\chipNZ214> sda cd
Logging to C:\Users\fyl\dss_11042013_100431.log
V6R2014x
```

```
Which development area would you like to work with?
[1] (Chip-NZ214) documenter-1_rmsith
[2] (Chip-QR2) verifier-1_thopkins
[3] (Chip-NZ214) developer-1_rsmith
[E] <EXIT sda>
Select the number preceding the development area name or 'E' to exit
 [1-3,E]: 1

Synchronizing the local development with the server ...
Contacting host: serv1.ABCo.com:2164 ...
Synchronization complete

Which tool would you like to launch?
[1] Authoring Tool
[2] DesSync
[E] <EXIT sda>
Select the number preceding the tool name or 'E' to exit (1-2,E): 2

c:\workspaces\chipNZ214>
```

Running sda cd in non-interactive mode

```
This example specifies the area and tool and the -noupdate option.
Note that it does not enter interactive mode, nor does it attempt to
synchronize the development area.  This example automatically
launches the GUI tool, without requiring the -GUI option because of
the way the tool is defined.

C:\workspaces\chipNZ214> sda cd Chip-NZ214 DesSync -noupdate
Logging to C:\Users\fyl\dss_11042013_103110.log
V6R2014x
[The DesignSync Development Area Manager launches in separate window]
c:\workspaces\chipNZ214>
```

sda mk

sda mk Command

NAME

```
sda mk              - Make a new development area
```

DESCRIPTION

- Running in Interactive Mode
- Tips for Naming Your Development Area
- External Development Areas
- Notes for Modules-Based Development (Module-based)

- Note for File-Based Development (File-based)

```
This command creates a new development area in the specified location
and registers the development area with the development server
managing the development. The development server and the
development the area uses must already exist.  For more information
on defining a development server, see the DesignSync Data Manager
Administrator's Guide.  For more information on development areas,
see the Enterprise DesignSync Administrator's Guide.

The sda mk command performs the following sequence of actions.
 o Creates the development area directory, if necessary.

 o Sets up the environment by creating environment variables to point
   to the new development area. The environment variables are:
    * SYNC_DEVAREA_DIR - the new development area directory.
    * SYNC_DEVELOPMENT_DIR - the top-level of the development instance
      directory.
    * SYNC_PROJECT_CFGDIR - the setting for the assignment associated
      with the development area.
 o Populates the development area with the development's data using
   the development URL from the development instance definition; the
   selector from the assignment associated with the development area;
   the version of DesignSync tools specified with the assignment; and
   any settings specified in the setting for the assignment, for
   example, the fetch state setting.  The development data is
   populated into a sub-directory of the development area named by
   using the leaf name of the containing server data. For more
   information see the appropriate note for your usage model.

   Note: For Windows development areas, the fetch state is
   automatically set to -get mode (Fetch Unlocked Copies).

Note: Server access may require a username and password.  If your
password for the server is not already saved by the client, you may
be prompted to enter it in order to access the server data. For more
information, see the notes section.

For information on defining a development server, see the DesignSync
Data Manager Administrator's Guide.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.
```

Running in Interactive Mode

```
Running sda mk with no arguments causes the command to enter the
interactive mode. In interactive mode, you are prompted for the
command arguments and options needed.

If you specify ambiguous or incomplete arguments, sda mk will enter
interactive mode only to resolve the unspecified or ambiguous
arguments.
```

Any arguments that are specified with the -gui command option will be passed to the GUI and the appropriate fields will be pre-filled or selected on the "Make Area" tab.

Tips for Naming Your Development Area

A development area name must start with an alphanumeric character and be composed of alphanumeric characters, including dot (.), dash (-), or underscore (__). Development names must be unique within a development server. Development area names must be unique for a development instance.

The command provides a unique default development area name in the following format:
<Assignment>-<count>_<creator>
Where:
<assignment> corresponds to the assignment selected previously, or entered with the -assignment option.
<count> is the next available number, starting from 1, of areas created. This is used to ensure the uniqueness of the name.
<creator> Username of the creator of the development area.

For example, User rsmith creating the first development area for the assignment Developer, has a default development area name of "Developer-1_rmsith"

Note: Development areas are checked for uniqueness in the name/instance pair. You cannot have to development areas for the same instance using the same name. You can have two development areas with the same name if they are for different development instances.

External Development Areas

An external development area is a development area whose physical presence is on a different network from the development server that it is associated with. External development areas are only allowed if the "Allow External Development Areas" parameter from the development definition on the development server is set to TRUE.

When a user creates a new development area, the sda mk command looks up the development instance path from the development definition on the development server. If the sda mk is run on a different network and can't find the development instance path, the command knows to create an external development area. The command then verifies that there is a local development instance directory for the development to host the new development area by checking for the existence of the directory located in the "External Path" parameter of the development definition on the development server. If this directory does not already exist, the command creates it.

The external development directory is similar in structure to the development instance created locally by the development server. The data replication root directory is replaced by a simple file cache directory. None of the external development's directory hierarchy is mirrored and no data is pushed to this directory directly from the development server. This is simply a local copy.

If the external development directory does already exist, its local development definition and the setting for the selected assignment is updated.

After the external development directory is in place and up to date, the normal development area creation procedure continues with the creation of the relevant environment variables and the data population.

Notes for Modules-Based Development (Module-based)

If the development data is managed as a module, the development area directory is the workspace root directory and the module data is populated into a sub-directory with the module's name. This allows for the main module to contain hrefs to peer modules which when populated recursively show up as peer subdirectories at the same level as the root module's base directory.

Server authentication for Windows systems using modules requires that the server be listed as a development server or pre-authenticated by saving the username/password for the server (using the password -save command). If the server is not authenticated, the development area is created, but the data is not populated.

Note for File-Based Development (File-based)

If the development data is file based, the data is populated into a sub-directory of the development area directory with the leaf name of the development URL, which is the directory name of the directory holding the data on the server. This structure allows the user to place unmanaged, peer, or derived data in the development area outside the data's directory cone.

Server authentication for Windows systems using file-based methodology requires that any servers referenced (by REFERENCE statements in sync_project.txt files) are pre-authenticated by saving the username/password for the server in order to populate the referenced data. To pre-authenticate your server, use the password -save command to save the username/password for the server.

SYNOPSIS

```
sda mk [<area_name> [<dev_name>]] [-assignment <assignment>] [-gui]
       [-path <path>] [-shared]
```

ARGUMENTS

- Area Name
- Development Name

Area Name

```
<area_name>     The new area name for the development.
                If no area name is specified, and the command is not
                run interactively, DesignSync uses the default name,
                in the format:
                <Assignment>-<count>_<creator>
                Where:
                <assignment> corresponds to the assignment
                  selected previously, or entered with the -assignment
                  option.
                 <count> is the next available number, starting from
                  1, of areas created. This is used to ensure the
                  uniqueness of the name.
                 <creator> User name of the creator of the
                 development area.
```

Development Name

```
<dev_name>      The development name of the DesignSync development
                instance to which the new development area is
                associated. The development must already exist.
```

OPTIONS

- -assignment
- -gui
- -path
- -shared

-assignment

```
-assignment     Specifies an assignment from a predefined list of
<assignment>   available assignments for the development. The
                assignment can be used to specify a module view or a
                different selector, one other than the default
                defined with the development, for the populate. If no
                assignment is specified, the "<Default>" assignment
```

```
                is assumed. The assignment determines the settings
                associated with the development area.
```

-gui

```
 -gui             Starts the sda graphical user interface mode
                  with the "Make Area," tab selected.
```

-path

```
 -path <path>     Specifies the area directory; the local
                  directory path where the development data will be
                 populated. If the directory already contains managed
                 data, the URL and selector of the data already
                 fetched into the directory must match the URL and
                 selector of the development combined with the
                 assignment.

                  Note: If you specify this option and the "Allow
                 user-defined development area paths" parameter is
                 set to FALSE, the command exits with an error.
```

-shared

```
 -shared          Designates the development area as a shared
                  development area. Shared development areas can be
                 joined by other users. All users of a shared
                 development area conduct their work in the same
                 development area directory.
```

RETURN VALUE

```
 This command does not return any TCL values.  The command output displays
 information about success or failure of the command and status messages.
```

SEE ALSO

```
 sda cd, sda gui, sda join, sda ls, sda rm, replicate
```

EXAMPLES

- Example of Running sda mk in Interactive Mode

Example of Running sda mk in Interactive Mode

```
$> sda mk
Logging to C:\home\rsmith\logs\dss_03132014_103149.log
V6R2019x
Contacting host: serv1.ABCo.com:2647 ...

Which development would you like to create a development area for?
[1] Chip-NZ8
[2] Chip-QR2
[3] ROM-NZx
Select the number preceding the development name or 'E' to exit (1-3): 1

Which assignment will be assigned to this development area?
[1] developer
[2] documenter
[3] verifier
Select the number preceding the assignment or 'E' to exit (1-3): 2

Please specify the name for the new development area
[documenter-1_rsmith]:

Please specify the path for the development area directory
[c:\Developments\rsmith\documenter-1_rsmith]:
C:\User\rsmith\DevAreas\nz8ChipDev

Should this be a shared development area (y/n) [n]:

The development area 'nz8ChipDev' for development 'Chip-NZ8' has been
created at c:\User\rsmith\DevAreas\nz8ChipDev
```

Exclude from Workspace

exclude

exclude Command

NAME

```
exclude             -  Commands for excluding objects from operations
```

DESCRIPTION

```
The exclude command allow you to control which unmanaged objects are
automatically excluded from check in or add operations on a per
directory basis.

Using the exclude commands, you can add, remove, or display the
glob-style exclusion patterns.  The exclusions are stored in one or
```

```
more syncexclude files.

Note: These exclusions are only applicable to unmanaged files.  If a
file is managed by the SyncServer, and you wish to exclude it from an
operation, such as populate, ci, or tag, you must use exclude lists
or filters (for example "populate -exclude *.doc").

The exclude files can be maintained either using these commands, a
graphical interface in the DesSync client, or by manually editing the
exclude file.  For more information on the files, the file format,
and using the various interfaces, see the DesignSync User's Guide:
Working with Exclude Files.
```

SYNOPSIS

```
exclude <exclude_command> [<exclude_command_options>]

Usage: exclude [add | list | remove]
```

ARGUMENTS

```
See individual commands.
```

OPTIONS

```
See individual commands.
```

RETURN VALUE

```
See individual commands.
```

SEE ALSO

```
ci
```

EXAMPLES

```
See individual commands.
```

exclude add

exclude add Command

NAME

exclude add - Add objects to exclude from operations

DESCRIPTION

This command appends the supplied pattern(s) to the end of the specified .syncexclude[*] file. If the specified file doesn't already exist, DesignSync will create it and place the supplied pattern(s) in it. Exclusions are processed in the order they appear in the file. You can edit the file to adjust the positioning of the exclusions or add an exclusion pattern with a higher priority.

Specify the pattern in one of the following forms:
-<pattern>
+<pattern>

When you use the "-<pattern>" form, you exclude objects that match the specified pattern at the folder level.

When you use the "- .../<patern>" form, you exclude objects that match the specified pattern at the folder level and any subfolders.

When you use the "+[.../]<pattern>" you create an exception to a previously excluded pattern. An example of using an exclude with an exception might be excluding all .doc files unless they're in the documentation subdirectory. So in the base-level .syncexclude, you could have this:

Exclude all doc files -".../*.doc"
and in a .syncexclude file within the documentation directory, you could have this:

+".../*.doc"

Any other sub-folders of the base folder would inherit excepting the unmanaged .doc files from revision control operations. The documentation directory and any subfolders of the documentation directory would allow .doc files to be included in revision control operations.

Note: Any changes to exclude files affect only unmanaged files. If a managed object matches the pattern, it remains unaffected. To exclude managed files, you must use -exclude or -filter, or an exclude list, as applicable. For more information on other types of exclusions, see the DesignSync User's Guide.

You must have write permissions in order to create or append to the file.

This command supports the command defaults system.

SYNOPSIS

```
exclude add <argument> [--] <pattern>[ <pattern>...]
```

ARGUMENTS

File Path Argument

```
<FilePath>    The path and name of the .syncexclude file. All
              .syncexclude files must begin with ".syncexclude" but
              can contain an extension which must begin with a "."
              character. For example, you could create a .syncexclude
              file that contains the module name, such as
              ".syncexclude.Chip." This allows you to include
              multiple .syncexclude files in the same directory. If
              the file extension does not being with a period, ".",
              it will not be understood by the system as a
              .syncexclude file.

              If the specified file does not exist, DesignSync
              automatically creates it.  If you do not have write
              permissions to create or modify the file, the command
              fails with an appropriate error.
```

Folder Path Argument

```
<FolderPath>   The path to the location of the .syncexclude file(s).
               If there are multiple .syncexclude files within the
               directory, the pattern is added to all of the
               .syncexclude files.

               The command does not operate in a folder recursive
               manner.  Only files at the specified directory level are
               updated.

               If there is no .syncexclude file in that folder,
               DesignSync automatically creates a new file called
               .syncexclude.  If you do not have write permissions to
               create or modify the file, the command fails with an
               appropriate error.
```

OPTIONS

- --

--

```
--              Indicates that the command should stop looking for
                command options. Use this option when the pattern
                supplied to the command begins with a dash (-).
```

PATTERN

- Pattern for Exclude

Pattern for Exclude

```
<pattern>       Specifies a space-separated list of patterns that
[...<pattern>] exclude or include unmanaged objects (collections,
                folders, or files) from check in or add
                operations, which would change the object from an
                unmanaged to a versionable object.

                Specify any pattern to exclude from operations that
                create managed objects or display unmanaged
                objects. Wildcards are allowed. Any patterns that end
                in foward-slash (/) apply to the folder and any
                files within the folder.  Do not use the backslash (\)
                character as a folder indicator. For specific usage
                information, see the Examples.
```

RETURN VALUE

```
No TCL value is returned.  If the command succeeds, DesignSync
displays a success message.  If the command fails, DesignSync
displays a message to explain the failure.
```

SEE ALSO

```
ci, exclude list, exclude remove, ls
```

EXAMPLES

- Example Showing Adding an Exclusion to the Exclude File
- Example Showing Adding an Folder-Based Exclude

Example Showing Adding an Exclusion to the Exclude File

```
This example excludes all unmanaged objects that end with a .log
suffix from revision control operations, such as ci.

Note: Because this is excluding a pattern, it requires the "--"
option to indicate that the next "-" is associated with the pattern,
not indicating an option.


dss> exclude add . -- -*.log
```

Example Showing Adding an Folder-Based Exclude

```
This example excludes all unmanaged objects in a folder that matches
the specified pattern. In this example, we have a directory
structure like this:
 rom
  doc
   rom.doc
   rom.fm
   rom.pdf
   rom.log
  log
   generatelog.log
   errorlog.log

Using our previous example, we have at the rom folder level a
.syncexclude that contains *.log.  But the log files within the .log
directory are files that should be checked in.  This plus exception
crteated in the same file allows the .log folder and all files within
to be operated on.

dss> exclude add . -- +../log/
```

exclude list

exclude list Command

NAME

```
exclude list         -  Show object patterns excluded from operations
```

DESCRIPTION

```
This command shows the contents of the exclude list files, allowing
you to see which patterns are excluded or included by the files in
the directory or .syncexclude file specified.
```

```
The command can display in either text or Tcl list form, to allow either
for easy viewing or additional processing.

This command supports the command defaults system.
```

SYNOPSIS

```
exclude list [-format text|list] <path>
```

ARGUMENTS

- File Path Argument
- Folder Path Argument

File Path Argument

```
<FilePath>    The path and name of the .syncexclude file.
```

Folder Path Argument

```
<FolderPath>   The path to the location of the .syncexclude file(s).
               If there are multiple .syncexclude files within the
               directory, the list of patterns for all the
               .syncexclude files within the directory are returned
               in the order in which they are processed.

               The command does not operate in a folder recursive
               manner. Only files at the specified directory level
               and higher in the folder hierarchy; back to the
               workspace root folder, are displayed
```

OPTIONS

- -format

-format

```
-format list|text  Determines the format of the output.
                   Valid values are:
                   o text   Display a text table with headers and
                            columns.  Objects are shown in processing
                           order.

                   o list   Tcl list structure, designed for further
                            processing, and for easy conversion to a
```

```
Tcl array structure. (Default) This means
that it is a list structure in name-value
pair format. The structure is:
{
   <path> <pattern>
   ...
}
```

RETURN VALUE

```
Empty string if -format value is text. Tcl list if the -format value
is list.
```

SEE ALSO

```
exclude add, exclude remove
```

EXAMPLES

Example Showing Listing the Exclusions in text format

```
This example shows the contents of a .syncexclude list in text
format. This .syncexclude file removes .log and .doc and includes
.readme, which was removed by a higher level .syncexclude.


dss> exclude list -format text
 File                                        Rule
 ----                                        ----
 C:/home/workspaces/Chip-ZN32/.syncexclude   -*.log
 C:/home/workspaces/Chip-ZN32/.syncexclude   -*.doc
 C:/home/workspaces/Chip-ZN32/.syncexclude   +*.readme
```

Example Showing Listing the Exclusions in List Format

```
This example shows the contents of a .syncexclude list in text
format. This .syncexclude file removes .log and .doc and includes
.readme, which was removed by a higher level .syncexclude.

dss> exclude list
{C:/home/workspaces/Chip-ZN32/.syncexclude -*.log}
{C:/home/workspaces/Chip-ZN32/.syncexclude -*.doc}
{C:/home/workspaces/Chip-ZN32/.syncexclude +*.readme}
```

exclude remove

exclude remove Command

NAME

```
exclude remove          -  Remove objects from being excluded
```

DESCRIPTION

```
This command searches all the specified .syncexclude files and removes
all occurrences of the specified pattern(s). The pattern specified
must exactly match the pattern in the .syncexclude file(s). If the
pattern uses wildcards in the .syncexclude file, you must use the
same wildcard pattern when specifying its removal.  Also, a wildcard
that, if processed, would match the pattern, does not remove an
entry.  For example, if the pattern in the file was:
 -dss*.log
specifying this pattern:
 -*.log
does not remove the pattern from the syncexclude file because it is
not an exact match.

To view the list of patterns in the file, so you can correctly match
the exclude pattern to remove it, you can use the exclude list
command.

You must have read and write access to the .syncexclude files and
directory.

This command supports the command defaults system.
```

SYNOPSIS

```
exclude remove <path> [--] <pattern>{<pattern>...]
```

ARGUMENTS

- File Path Argument
- Folder Path Argument

File Path Argument

```
<FilePath>    The path and name of the .syncexclude file.
```

Folder Path Argument

```
<FolderPath>   The path to the location of the .syncexclude file(s).
               If there are multiple .syncexclude files within the
               directory, the pattern is removed from all of the
               .syncexclude files that contain that pattern.

               The command does not operate in a folder recursive
               manner.  Only files at the specified directory level are
               updated.
```

OPTIONS

- --

--

```
--             Indicates that the command should stop looking for
               command options. Use this option when the pattern
               supplied to the command begins with a dash (-).
```

PATTERN

- Pattern for Exclude

Pattern for Exclude

```
<pattern>      Specifies a space-separated list of patterns that
[...<pattern>] must exactly match a pattern specified in the
               .syncexclude files affected by the command.
```

RETURN VALUE

```
Returns the number of removals. If there are no patterns that match
the specified pattern, the removal number is zero "0". If the command
fails, returns an error explaining the failure.
```

SEE ALSO

```
exclude add, exclude list
```

EXAMPLES

- Example Showing Removing an Exclusion from the Exclude File

Example Showing Removing an Exclusion from the Exclude File

```
This example shows removing one of the exclusions created in an
exclude add example.
dss> exclude remove . -- -*.log
 2
```

populate

populate Command

NAME

```
populate            - Fetches or updates specified objects
```

DESCRIPTION

- Object States
- How Populate Handles Selectors
- Populate Log
- How Populate Handles Collections with Local Versions
- Populating Module Objects (Module-based)
- Setting up Your Workspace (Module-based)
- How Populate Handles Module Snapshots (Module-based)
- How Populate Handles Module Views (Module-based)
- Resolving Module Conflicts with Populate (Module-based)
- Module Cache (Module-based)
- External Module Support (Module-based)
- Populating Modules Recursively (Module-based)
- Module Version Updating (Module-based)
- Incremental Versus Full Populate (Module-based)
- How Populate Handles Moved and Removed Module Members (Module-based)
- Merging Across Branches (Module-based)
- Understanding the Output (Module-based)
- Forcing, Replacing, and Non-Replacing Modes (Module-based)
- Interacting with Legacy Modules (Legacy-based)
- Incremental Versus Full Populate (Legacy-based)
- Setting up Your Workspace (File-based)

```
This command fetches the specified objects from the server
into your current workspace folder or a folder you specify
with the -path option.

Typically, you create your work area, or workspace, and perform your
first populate, an initial populate, as a full populate.  Once your
work area is populated, you can use the populate, co, and ci commands
to selectively check out and check in specific objects. You should
also populate periodically to update your work area with newly
managed objects, as well as newer versions of objects you have
locally.

Populate is used to create or update the objects in your
workspace. Populate features many ways to control the data brought
into your workspace. Because of the complexity of the populate
features, the description section is divided into sections that
detail the major features and functionality of populate.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

This command supports the command defaults system.
```

Object States

```
Upon populating your workspace, DesignSync determines in what
state to leave the fetched objects in your work area:
  1. DesignSync obeys the state option (-get, -lock, -share,
     -mirror, -reference) specified on the command line.
  2. If no state option is specified, DesignSync uses the default
     fetch state as specified by your project leader. See
     the "fetch preference" help topic for more information.
  3. If a default fetch state is not defined, the default
     behavior for 'populate' is -get.

Important: For both incremental and full populate operations,
DesignSync changes the state of only those objects that need
updating.  DesignSync does not change the state of up-to-date
objects during the populate operation.

The following methods let you override the default behavior
to change the states of all objects during a populate operation:
  o To change the state of up-to-date objects during a populate,
    use the -unifystate option.  To change the state of all
```

objects that need an update as well as up-to-date and locally modified objects, use -unifystate with the -force option.
- Unlocked locally modified objects are not overwritten unless you specify -force. For example, if you modify a fetched file, then execute a 'populate -share' command, your locally modified file is not replaced by a link to a file in the cache unless you also specify -force. Locked files are not overwritten by the -force option.
- To make populating with links to the mirror a fast operation, links are created only if no object (locally modified or not) or link already exists in your work area. You must specify -unifystate to change the state of existing objects and links in this case. Use -force, as well, to overwrite locally modified objects that are not locked and to remove objects that are not in the current configuration.

Note: If the object is designated as uncachable, attempts to place objects in the cache (populate -mirror; populate -share) will automatically populates the workspace with unlocked copies (-keep mode). For more information on cachability, see the "caching" commands.

How Populate Handles Selectors

DesignSync determines what versions of objects to populate as follows:

1. DesignSync obeys the selector list specified by the -version option.
2. If -version is not specified, DesignSync uses the persistent selector list of the top-level folder being populated. The default persistent selector is 'Trunk', in which case DesignSync checks out the Latest versions from Trunk.

Notes:
- If you specify a selector or a selector list for the populate operation using the -version option and the selector does not exactly match the workspace selector, an incremental populate is no longer valid. In this case, DesignSync performs a full populate even if the -incremental option is specified. See "Incremental Versus Full Populate" above for more information.

Important: The persistent selector lists of individual managed objects (files or collections) and subfolders are not obeyed by the 'populate -recursive' operation.

- A 'populate -recursive' command without the -version option populates a work area based on the persistent selector list of the top-level folder you are populating, skipping any subfolder or managed object that has a persistent selector list that differs from the top-level folder. You must issue the populate command separately for any skipped subfolder.

- A 'populate -recursive -version <selectorList>' command uses the specified selector list and ignores all persistent selector lists. In the case of '-version Latest', the persistent selector list of the top-level folder being populated is augmented with 'Latest' and that augmented persistent selector list is used for the populate operation.

The supported DesignSync use models (single-branch development, project branching, and auto-branching) assume that persistent selector lists across a work area are consistent. Use caution when using commands that leave you with inconsistent local metadata, such as using 'setselector' or 'mkbranch' on individual objects.

See the "selectors" help topic for details on selectors, selector lists, and persistent selector lists. For more information about how the -version switch is managed, see the -version in OPTIONS.

Populate Log

Because populate operations can be long and complex, you may want to specify a log file to contain only the output of the populate command to store for later reference.

You can specify the log file on an as needed basis using the -log option or by setting a log file name using the command defaults system. If the log file specified does not exist, DesignSync creates it before it begins the populate command processing. If the log file does exist, DesignSync appends the new populate information to the file.

Tip: If you set a default log value for populate, check the file size periodically and, if the file is getting too large to use comfortably, rename the file to save the information, or remove the file if you no longer need it.

Notes:

- If a log file is defined in the command defaults system and two users run populate simultaneously, the populate output may become interlaced in the log file.

- Regardless of whether you create a populate log, the DesignSync client log file contains the output of the populate command along with all the other commands typed into the DesignSync client session.

How Populate Handles Collections with Local Versions

For collection objects that have local versions (for example, custom generic collections), the populate operation handles local

versions in the following way.

When you populate a folder containing a collection object, the populate operation removes from your workspace any local version of the object that is unmodified. (Because these local versions exist in the vault, you can refetch them.) The operation then fetches from the vault the specified collection object (with the local version number it had at the time of checkin).

If the current local version in your workspace is modified, the populate operation fails unless you specify 'co -force'. (The -force option lets the local version with the modified data be replaced with the local version of the object you are checking out.) Note: The current local version is the one with the highest local version number. DesignSync considers a local version to be modified if it contains modified members or if it is not the local version originally fetched from the vault when the collection object was checked out or populated to your workspace.

The -savelocal option tells the populate operation what to do with local versions in your workspace other than a current local version that is modified. For information, see OPTIONS.

Populating Module Objects (Module-based)

The populate command recognizes and fetches hierarchical module structure. These modules are data that represent a level of the design hierarchy. Such data includes objects or an entire vault folder hierarchy of objects managed in DesignSync, as well as hierarchical references to other modules. These modules can be stored on other SyncServers. For more information about modules, see DesignSync Data Manager User's Guide: "What is a Module?".

Important: You must use the populate command rather than the co command when fetching modules or module objects. The co command does not support modules.

To specify a module for an initial populate, you must specify its server URL, in the following format:
sync://<machine>:<port>/Modules/<category>/<module_name>[;<selector>]

DesignSync looks for an existing workspace root. If no workspace root exists and the registry key AllowAutoRootCreation is enabled, DesignSync automatically creates the workspace root based on the value set for DefaultAutoRoot path. If there is no existing workspace root path and DesignSync cannot create one, the populate fails. Workspace root path settings are in the DesignSync registry.

During the initial populate, DesignSync performs an implicit setvault. If necessary, DesignSync also creates a workspace folder for the module. For subsequent populates, you do not have to specify the server URL for the module; you can populate the module by specifying just the module name or the module instance name if your current directory is within the workspace root (see the setroot command

help), or using the full workspace address which is "<module base directory>/<module instance name>".

If a top-level module (a module that is not hierarchically subordinate to another module populated in the workspace) is populated with the -version option, the persistent selector for the workspace is changed to the version specified.

Overlapping of modules is supported. You use the -modulecontext option to indicate which module to populate if more than one module exists in the current directory (or that specified with the -path option). If no -modulecontext option is specified, all appropriate module objects from the candidate modules are populated.

If a file is a member of both overlapping modules, a populate clash occurs. In this case, the first module to populate the file 'wins'. A subsequent attempt by an overlapping module to populate the same file fails.

Two different versions of the same module cannot share the same base directory. However, you can populate two versions of the same module side by side.

Notes:
- Mirrors are not supported with module objects; you get an error if you use the -mirror option.
- If a module member is checked out with a lock, the locker keyword is not expanded with the locker name.
- You can use the -mcachemode, -mcachepaths, or -noreplace options only when populating a directory that is part of a module or a legacy module.
- After the upgrade command has been used to convert legacy modules to a module, fetch each new module to an empty work area. The upgrade command does not upgrade existing work areas.

Setting up Your Workspace (Module-based)

Before you can use populate to maintain your workspace, you must set up your workspace.

Note: The Workspace Wizard from the DesignSync graphical user interface simplifies the task of setting up a work area by taking you through a step-by-step process.

The typical steps when you set up a new work area are:

1. Create the folder for your workspace, if it does not already exist.

2. Populate the work area with the specified design objects from the vault. Populate determines the set of versions from the persistent selector list or from the -version option, if applicable. Apply the -recursive option to create a local hierarchy that matches the vault's hierarchy. Without

-recursive, populate only fetches the specified objects.

How Populate Handles Module Snapshots (Module-based)

A module snapshot is a set of meaningful tagged module objects. The content and structure of a module snapshot is frozen to preserve important configurations. After the module snapshot has been created using the tag command, you can populate the snapshot into a local workspace for viewing, testing, or integrating into other work.

When you populate a module snapshot as a fixed workspace for viewing or testing, you use the snapshot tag as a selector. This can be either the full snapshot branch and version name or the simple tag name. When you populate a snapshot module, you can update tags on module members or hrefs within your workspace, but cannot checkin any content or other structural changes to the module members or the module.

When you populate a module snapshot to integrate with other work, you populate using a comma separated list of selectors ending with a "main" selector. This populates from the main selector first and replaces any matching objects with the member objects from the selectors in the selector list.

This results in a workspace that uses the main selector as the base and the destination for any checkins, but some or all of the module member objects from the snapshot workspaces. For example, specifying the following version to populate:
Beta,Alpha,Trunk:Latest

The populate command creates a module manifest from the main selector, Trunk:Latest, and overlays that with the contents of the Alpha version, and then the Beta version. The final manifest is then sent to the client. The server uses the natural path of the objects and the uuid to determine which module members to replace.

When hierarchical references are populated as part of the operation, the hierarchical reference versions come from the main selector list, not from the specified module snapshots.

When the hierarchical references are populated recursively during the initial populate using a selector list, the module members within the populated submodules are also populated with the selector list. If hierarchical references are not populated recursively during the initial populate using a selector list, they will not overlay member items from the selector list on subsequent populates.

Notes:

- If the "main" selector list is a snapshot branch, or a static selector of any type, you will not be able to check in any changes from the workspace.

- When populating a selector list, the module member objects in the specified snapshot are populated instead of the objects in the

main selector. Populate will never attempt to merge the members. If you want to merge data from a module snapshot into your workspace, you will not use a selector list, but populate your snapshot with the -merge and -overlay options into a workspace that has the default selector defined as the desired destination for checkin.

- Any hierarchical references that are defined as a static module version indicated by the selector on the href will not inherit any the selector list, even if the initial populate specifies using the selector list recursively.

How Populate Handles Module Views (Module-based)

A module view is a defined subset of module members and hierarchical references that have significance as a unit. The module view definition is stored on the server with a unique module view name. During populate, you can specify the view name to restrict the populate operation to only those members in the view. You can populate using more than one view.

Note: During initial populate, if you specify a view, the view specified persists in the workspace.

The populate operation builds the list of module members and hierarchical references (if run recursively) to populate by first looking at the specified view(s) on the specified module and selector. After building this aggregate set of data, DesignSync applies the filtering rules from the -filter, -hreffilter and -exclude options to determine what objects to populate into the workspace.

On an initial populate, the module view name or names list provided is propagated through the hierarchy and applied to all fetched modules. The module view name or names list is also saved, or persisted in the workspace metadata so that all subsequent populates use the same view. The documentation refers to a view saved in the metadata as a "persistent module view" because, like a persistent selector, it persists through subsequent populates rather than needing to be specified with each command.

If a persistent module view has been set on a module instance in a workspace any sub-modules subsequently populated use the persistent module view already defined by default.

Note: You can set or clear a persistent selector by using the setview command.

Resolving Module Conflicts with Populate (Module-based)

DesignSync provides the ability to define an overriding hierarchical reference to be used in cases where submodule references point to

different versions of the same object. This can be used in both a peer-to-peer or hierarchical cone structure. In a peer-to-peer structure, it can be used to resolve conflicts and determine which version of the sub-module to populate into workspace.

For example, a module called TOP with hrefs to sub-modules:
ROM@1.23 -relpath ../ROM
COM@1.15 -relpath ../COM

where ROM and COM both contain an href to a common libraries directory, but to different versions:
ROM -> LIB@1.3 -relpath ../LIB
COM -> LIB@1.5 -relpath ../LIB

Working in a peer-based structure, where your modules are populated in a flat directory setting, your workspace may look something like this:
/home/workspace/TOP
/home/workspace/ROM
/home/workspace/COM
/home/workspace/LIB

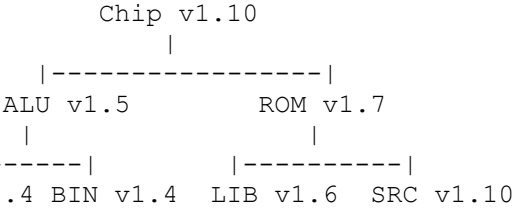
DesignSync may experience a conflict determining what version of LIB (1.3 or 1.5, as referenced in the hierarchy)to populate in the peer directory /home/workspace/LIB.

If an href is placed higher in the peer structure, however; it will become the overriding href. So, for example, if you add an href for TOP to LIB, as shown:

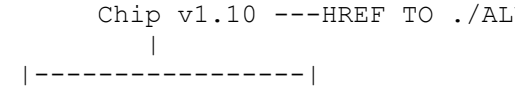
```
 TOP -> ROM@1.23 -relpath ../ROM
     -> COM@1.15 -relpath ../COM
     -> LIB@1.5 -relpath ../LIB
```

When you populate the TOP workspace recursively into /home/workspace/TOP, DesignSync populates the LIB directory with the 1.5 version, eliminating the guesswork.

In a cone structure, it can be used to substitute a submodule version without modifying the hierarchy or branching the sub-module to update an href version. For example:



If rather than branching ALU and updating the hierarchical reference to LIB, you add an href to the desired version of LIB at a higher level, for example, Chip, then that version of LIB will replace the lower level version with the same relpath when populated.



```
      ALU v1.5         ROM v1.7
        |                 |
  |---------|         |----------|
LIB v1.8 BIN v1.4  LIB v1.6 SRC v1.10
```

Notes:

o The relpath of the hierarchical reference is what's used to determine which sub-module is replaced.

o In order for the overriding href to be used by the system, you must populate recursively from the highest level module containing the override href. For example, if you were to populate either of the above examples at the ROM level, the ROM href is the one that is used to determine what submodule is populated; not the higher-level module.

Module Cache (Module-based)

A module cache (mcache) can be thought of as a shared workspace. The populate command works with both module and legacy module mcaches. A module mcache contain modules while a legacy mcache contains only legacy releases.

To create a module cache, team leaders should create a workspace and populate it with modules and or legacy modules using the -share option. This becomes the mcache directory. Usually a team leader creates the mcache for team members to access over the LAN. The mcache should be writable only by the team leader. Team members should need permission to read the data, link to and copy the module or legacy module in the mcache.

Note: The module cache must be the workspace root directory.

An mcache is manually administrated. Modules and legacy modules can be fetched as needed. You can have multiple modules in the mcache.

o You can have full copies of all the modules in an mcache.
o If you use -share option to populate an mcache, it allows you to keep full copies of the DIFFERENCES between versions by populating the mcache from the DesignSync cache which stores the files.

Note: Only statically fetched modules can be fetched from an mcache during populate.
Only released configurations can be fetched from an mcache during populate.

Since multiple modules can have the same base directory or have the same directory at various levels, it can cause confusion for mcache links and can even cause circular or inconsistent links. To keep the contents of a mcache consistent, an mcache link to an mcached directory containing modules are created for only one module version.

An mcache can either be for modules or legacy modules, not both. A

module can have hierarchical references to legacy modules, resulting in the legacy modules being populated to the module mcache. These legacy modules are ignored when creating mcache links or copies.

The -mcachemode copy option is ignored for modules. You can, however, get the contents of a module from the LAN if your team lead fetches the modules from the server into the mcache using the -share option. This forces the module contents to get fetched into the DesignSync cache (different from an mcache). Symlinks are created in the mcache to point to these files in the DesignSync cache.

If you specify -mcachemode copy to get full copies of a module's contents from the mcache, the populate operation automatically switches the command to use the default '-from local' mode to fetch the files.

To use a module mcache the root directory of the mcache must be provided in the -mcachepaths option or the mcache paths registry setting. This root directory contains the metadata identifying the base directories of all module cache. See the section on -mcachepaths for more information.

Note: If a module, module category, the Module area or server is designated uncachable, it cannot be stored in an mcache. If a module has already been populated into a cache and is then designated as uncachable, the module cache is not automatically removed.

External Module Support (Module-based)

DesignSync supports populating an external module, an object or set of objects managed by a different change management system, within a module hierarchy. Using an external module in a DesignSync hierarchy allows you to manage code dependencies between module objects in DesignSync and files checked in to other change management systems.

Within a parent module, you add an href that refers to an external module. The external module reference contains the name of an external module interface. The external module interface, provided by an administrator, defines a procedure to populate the sub-module using an external change management system.

After creating the href to the external module, you populate it exactly as you would any other href, by specifying either the href name or the module instance name as the populate argument, or by populating the parent module with the -recursive option.

The external module must be part of a module hierarchy. You cannot create an external module as a top-level module. Once in the workspace, the module itself, or any subfolders, or objects within the module may be individually populated according to the external module interface definition.

Notes:
o The external module's directory structure cannot overlap with

any other module data.

- If an external module populate fails and the populate command was run with the -report brief option specified, you may not have enough information to determine where the failure occurred. If you rerun the populate with the -report brief mode, you can locate the referenced object within the module hierarchy.

Populating Modules Recursively (Module-based)

You can use populate to fetch entire modules or their members as follows:

- To fetch a single module without fetching its submodules, specify the workspace or server module and apply the populate command without the -recursive option.
  The command populates the module members without following hierarchical references (hrefs).

- To fetch all objects in an entire module hierarchy, specify the workspace or server module and use the populate command with the -recursive option.
  The command traverses the hierarchy in a module-centric fashion, populating all the objects in the module and following its hrefs to populate its referenced submodules.

- To fetch all objects in a folder, specify a folder name and apply the populate command without the -recursive option.
  The command fetches the objects in the folder, without following hrefs.

- To fetch all objects in a folder and its subfolders, specify a folder name and apply the populate command with the -recursive option.
  The command traverses the folders in a folder-centric fashion, populating the modified objects in the folder and its subfolders, but without following hrefs. To follow hrefs, you must specify a workspace or server module instead of a folder.

- To fetch all objects in a module or module hierarchy but restrict the fetch to a particular folder hierarchy, use the -modulecontext option to specify the module and provide the folder name.
  - Specify the -recursive option if the module hierarchy needs be traversed to fetch items from the sub-modules into that folder.
  - Specify -norecursive option to fetch only the items from the given module. Note that this operation is "module centric" and "folder recursive", in that all items in the module are fetched which belong to the given module or its sub-folders.
  - To restrict the operation to both a module and a single folder, use the -filter option to filter out items from sub folder.

Note: You cannot specify the -recursive option, if you are performing a cross-branch merge (with pop -merge -overlay) on a module.

When you fetch a module recursively, you update the module hierarchy. How that module hierarchy populates depends on the href mode specified, and the selector(s) specified within the href, the hreffilter string and possibly the populate selector for the selected module. For more information on how the module hierarchy is populated, see the "Module Hierarchy" topic in the ENOVIA Synchronicity DesignSync Data Manager User's Guide.

Note: If the "HrefModeChangeWithTopStaticSelector" registry key is enabled, and the selected module is a static version, the static version is saved as the persistent selector in populate. For more information about setting the "HrefModeChangeWithTopStaticSelector" registry key, see the ENOVIA Synchronicity DesignSync Data Manager Administrator's Guide.

Module Version Updating (Module-based)

The populate command updates the module version upon successfully fetching all members of the module. If the populate command is not completely successful, the fetched version number is not updated, as in the following scenarios:

- A module member cannot be fetched if the member is locally modified (and -force is not applied). In this case, the module is not fully populated, and the module version is not updated.

- A module member is not fetched if a -filter, -exclude, or -nonew option excludes it. In this case, the module is not fully populated, and the version number is not updated.

If you do not have the Latest complete module version due to one of these cases, you can still check in a module; the ci command auto-merges members so that the module is fully updated upon checkin. See the ci command for details.

You can use the showstatus command to detect whether a module has been fully populated. The showstatus command lists the module as 'Needs Update' if the Latest version has not been successfully fetched.

Unlike the cases where the module version is not updated, the module version is updated if a populate successfully updates the entire module, but fails to remove files that are no longer members of the module. If a member has been removed from the new module version, but the populate command cannot remove it from the workspace (because it is locally modified and -force was not applied), the workspace does contain the entire contents of the new module version, so the module version is updated.

Incremental Versus Full Populate (Module-based)

By default, the populate command attempts to perform an incremental populate which updates only those local objects whose corresponding vaults have changed. For modules, DesignSync tracks the members changed on the server and in the workspace and performs an incremental populate. Avoiding a full populate improves the speed of the populate; however, some circumstances make a full populate necessary. In the following cases, DesignSync automatically performs a full populate:

- If you are populating with a different configuration to that of the work area (having used setselector, setvault, 'populate -version', or 'co -version' to change a selector), DesignSync performs a full populate. For example, if your last full populate specified the VendorA_Mem configuration, but you now want VendorB_Mem files, then DesignSync automatically performs a full (nonincremental) populate. If the selector you specify resolves to the same exact selector as that of the work area, DesignSync does perform the incremental populate. In this case, the selectors must be an exact match; for example, a selector which resolves to 'Main' does not match 'Main:Latest'. If you are populating with a new configuration, consider using the -force option to remove objects of the previous configuration from your work area.

- If you have removed module data from the workspace with rmfile or rmfolder, DesignSync performs a full populate, refetching the removed files.

- If you use the -lock option, DesignSync performs a full populate.

- If you use the -unifystate option, DesignSync performs a full populate.

- If you perform a nonrecursive populate on a subfolder, all of the folders above the subfolder are invalidated for subsequent incremental populate operations. Incremental populate works by exploiting the fact that if a folder is up-to-date, all of its subfolders are also up-to-date, making it unnecessary to recurse into them. Because a recursive populate was not performed for the subfolder, DesignSync cannot ensure that its subfolders are up-to-date; thus, all incremental populates are invalidated up the hierarchy.

- If you perform a nonrecursive populate on a folder, DesignSync essentially runs a full populate rather than the default incremental populate. Your next populate is incremental from the last recursive populate. If you have not previously run a recursive populate, the subsequent populate is a full populate.

Note: If you are using a mirror (by specifying -mirror or having a default fetch state of Links to mirror), an incremental populate does not necessarily fetch new objects checked in, nor remove links to objects deleted by team members until after the mirror has been updated.

For the following cases, you should perform a full populate instead of an incremental populate:

- If you have excluded a folder by using the -exclude, -filter, or -noemptydirs option with the populate command, a subsequent incremental populate will not necessarily process the folder of the previously excluded object. DesignSync does not automatically perform a full populate in this case. To guarantee that previously excluded objects are fetched, specify the -full option for the subsequent populate operation.

- Specify a full populate to force data that has been manually removed, removed locally, or renamed locally to be fetched again from the server. If the file was renamed, you may have to specify the -force option as well.

  Also, specify a full populate if you have an unchanged, but out-of-date or out-of-sync version in your workspace to force DesignSync to fetch the up-to-date version of the object.

- If the system clock on the SyncServer machine where your vault is located is turned back (for example, to correct clock skew between machines), you must perform a full (nonincremental) populate to synchronize the client and server metadata.

- If you interrupt a populate operation (using Control-c, for example), you should use populate -full on your next populate of that folder.

The default populate mode is -incremental; however, your project leader can set this default using SyncAdmin.

If you are updating mirrors, use the -incremental option. If you specify the -full option, mirror updates can take a long time to complete.

Note: If you remove objects from the work area by using operating system commands rather than DesignSync commands, an incremental populate cannot fetch these objects. Perform a full populate (-full) or use the -unifystate option to fetch them.

How Populate Handles Moved and Removed Module Members (Module-based)

When you populate a module, DesignSync does not populate any module member that has been removed on the server. Existing module members in your local workspace that have been removed on the server are removed during a populate.

Module members that have been removed or moved locally, but those changes were not committed to server are preserved in the workspace unless populate is run with the -full and -force options which remove the local modifications (including the structural changes) and replace the workspace version with the server version.

Merging module members that have been removed or renamed is discussed in Merging Across Branches

Merging Across Branches (Module-based)

In multi-branch environments, you use the populate command to merge branches. In many cases, a new branch that is created is eventually merged back into the main development branch.

The branch being merged is populated into a workspace containing the destination branch using the populate command with the -merge and -overlay options. This type of merge is called "cross-branch merge."

As with all populate operations, cross-branch merging uses the filter and exclude filter lists set on the workspace, in the command defaults system,on the command line.

Note: Filtering on module workspaces is applied to the natural path of the module members. If a module member's natural path has changed, creating a situation where either the new location or the old location, but not both is excluded, the module member is included in the merge.

Important: When working with modules, you should lock your workspace branch before beginning a cross-branch merge. This reduces the risk of changes being committed by another user while you are merging the versions. After the merge has been completed, the changes have been reviewed and accepted, and the new module version created, unlock the branch to make it available for general use.

Merging includes two basic types of merging: file contents, and structural changes.

- File content merging:
  File content merging is applicable to all DesignSync objects including module members. DesignSync merges the contents of files with the same natural path to the best of its ability. If the files are binary files which cannot be merged, populate returns an error message.

- Structural change merging for Modules:
  Structural changes for modules are either committed when the module is checked in or can be individually committed. Structural changes for Modules include:

  - Removed objects - If an object is present in the local workspace, but has been removed on the merge version, it is marked with a metadata property to indicate that it was removed from the branch. If you want to remove it from the merged module version, you must manually remove the file from the workspace before creating the new module.

    If the object has been removed on the workspace, but:

* is present on the server at the same member version removed from the workspace, the object remains in the same state, and is removed from the server during the next checkin.

* is present on the server at a newer version or has been moved, or is on the overlay version, the new version is not merged into the workspace, and an error is returned stating there is new version. The version in the workspace remains in the removed state, but you will not able to check in the change until you resolve the merge conflict.

- Added objects - If an object is present in the merge version, but not in local workspace, it is added to the module and is checked into the module when the next checkin operation on the module or the module member is performed.

- Moved or Renamed objects - A moved (or renamed) object has a different natural path. Objects that have been moved on either the server or checked in from the workspace have been moved on the server. Objects that have been moved in the workspace, but have not been checked in are considered moved locally.

  If an object has been moved on the server, but not locally, the module member in the workspace retains the same name or location in the workspace, and a metadata property is added to the object to indicate the new path name. To determine what files have been moved, review the populate status information, log file, or run the ls command with the -merge rename option.

  If an object has been moved locally, and:

  * has been moved on the server to the same location, the merge operation is performed on the merged local version. Subsequent checkin checks in the merged file to the new location. If the content has changed, DesignSync will perform a content merge as well.

  * has been removed on the server, the new version is not merged into the workspace, and an error is returned by populate. new version. The version in the workspace remains in the moved state, but you will not able to check in the change until you resolve the merge conflict.

  * has been updated on the server, content changes are merged into the moved file, and subsequent checkin of the member moves the file on the server and updates the content.

  * has been moved on the server to a different location and updated, the content is merged, the workspace version remains in the same location in the workspace, and an error is logged in populate to alert you that the file has been moved on the server. In order to checkin, you must resolve the merge name conflict or checkin with the -skip option to move the file to name of the file in your local workspace.

  * and exists on the overlay version, the overlay version is not copied into the workspace, but a metadata property is placed on

```
      the local version to indicate that natural path of the object
      is different.  You can see a list of these differences by using
      ls -merged.

    Note: If a file marked as renamed is subsequently renamed again,
    or removed from the module, the metadata property indicating that
    the file was renamed by merge may persist.  To clear the
    property, perform the mvmember or remove command on the workspace
    object, or manually clear the property using the url rmprop
    command.

  - Added or Removed hierarchical references - Hierarchical reference
    changes cannot be merged.  You must manually adjust your
    hierarchical references.

After a cross-branch merge has been performed, you can view the
status of the affected files using the ls command with the -merged
<state> -report D options.  The -merged option allows you to restrict
the list to a particular type of merge operation (add, remove,
rename, all) and the -report D option shows you the current state of
the object in your workspace.  For more information, see the ls
command help.

When a merge is performed on a DesignSync object, a merge edge is
created automatically to maintain a list of the changes incorporated
by the merge.  This identifies a closer-common ancestor to provide
for quicker subsequent merges. When performing a cross-branch merge
on a module, however, you need to manually create the merge edge
after committing the selected changes.  For more information on
creating a merge edge, see the mkedge command.

For more information about merging, see the -merge and -overlay
options, and the DesignSync Data Manager User's Guide topic: "What Is
Merging?"

Notes:
o Auto-branching is not supported for modules; you cannot specify
  the auto-branching construct, auto(), for modules.
```

Understanding the Output (Module-based)

```
The populate command provides the option to specify the level of
information the command outputs during processing. The -report option
allows you to specify what type of information is displayed:

If you run the command with the -report brief option, the populate
command outputs a small amount of status information including, but
not limited to:
 o Failure messages.
 o Warning messages.
 o Version of each module processed as a result of a recursive
    populate.
 o Removal message for any hierarchical reference. removed as part of
   a recursive module populate.
```

- Informational messages concerning the status of the populate
- Success/failure/skip status

If you do not specify a value, or the command with the -normal option, the populate command outputs all the information presented with -report brief and the following additional information:

- Informational messages for objects that are updated by the populate operation.
- Messages for objects excluded from the operation (due to exclusion filters or explicit exclusions).
- For module data, also outputs information about all objects that are fetched.

If you run the command with the -report verbose option, the populate command outputs all the information presented with -report normal and the following additional information:

- Informational message for every object examined but not updated.
- For module data, also outputs information about all objects that are filtered.
- For module versions that have been swapped, output indicates when the selector of a swapped sub-module is being used.

If you run the command with the -report error option, the populate command outputs the following information:

- Failure messages.
- Warning messages.
- Success/failure/skip status messages.

Note: References to DesignSync Vault, IPGear Deliverables, or External modules do not have a module instance name to add to the object path. When running with the error report mode, if an object within a referenced DesignSync Vault, IPGear Deliverable, or External module fails, you may need to rerun the operation with the report -brief option to locate the referenced object within the module hierarchy.

Forcing, Replacing, and Non-Replacing Modes (Module-based)

You can use these three modes to specify how the populate command updates your work area:

- Forcing mode (specified with the -force option) synchronizes your work area with the incoming data, including locally modified objects. In this mode, the populate command updates the managed objects in your work area. It replaces or removes managed objects regardless of whether the objects have been locally modified and whether they are members of the module being fetched. Thus, forcing modifies your work area to match the set of module members being fetched. Note: The default -noforce option operates as if -replace has been specified.

- Replacing mode (specified with the -replace option) also synchronizes your work area with the incoming data, but without

affecting locally modified objects (the default behavior). For modules, the populate command updates managed members of the module that have not been locally modified. It also removes any unmodified managed objects that are not members of the module being fetched.

Replacing mode, unlike forcing mode, leaves intact managed objects that have been locally modified.

- Non-replacing mode (specified with the -noreplace option) is the least disruptive mode; this mode might require you to clean up the resulting work area data.

  In this mode, the populate command takes the incoming data and overlays it on top of the existing work area's data. It leaves intact both managed objects with local modifications and managed objects that are not members of the module being fetched. Thus, the work area essentially becomes a union of the data from the previous version and that of the module being fetched.

  Non-replacing mode, unlike forcing mode, leaves intact any objects that have been locally modified, and, unlike the replacing mode, leaves unmodified managed objects intact. See the -[no]replace option below for more details.

Notes:
- Unmanaged objects in your work area are not affected by any of these modes.
- The following are illegal combinations of options: -replace and -noforce, as well as inverse options, such as -replace and -noreplace.

Interacting with Legacy Modules (Legacy-based)

The general functionality provided by populate is provided for legacy modules by the hcm get command. The sections within populate that are specifically tagged for legacy modules refer to interactions with modules or files-based objects, when populate is used, or if populate is used on individual objects, not an entire legacy module configuration. For more information on updating legacy modules in your workspace, see the hcm get command.

Important: Legacy modules are modules generated prior to Developer Suite 5.0. The modern modules functionality provides significant improvements. You can update your legacy modules using the upgrade command.

Prior to Developer Suite 5.0, legacy modules were managed with module configurations. Modules no longer require "configurations". A configuration was a set of object versions sharing a common tag (for example, files of a version tagged 'Rel2.0' comprise the Release 2.0 configuration).

In ProjectSync, a configuration represents a state in the life-cycle of a project. It has an owner, team members. When associated with a DesignSync vault, the configuration has a selector list (typically a tag) identifying the versions of DesignSync data that are part of the configuration.
ProjectSync project and configuration information is stored in a sync_project.txt file that is located in the project folder.

When you populate based on a name that corresponds to a ProjectSync configuration, DesignSync uses the selector list (typically a tag name) associated with that ProjectSync configuration to identify the versions to be populated. This scenario is called configuration mapping.

Configuration mapping is used when a configuration name does not have the same meaning for all modules of a project. For example, a project's Alpha configuration may consist of the Gold configuration of one module, the Rel20 configuration of another, and several other modules whose design files are actually tagged Alpha. Configuration mapping lets you identify these different versions of design data with one configuration name.

When you populate a configuration-mapped folder (either directly or through a recursive populate operation) and the selector you specify is mapped, the persistent selector list for that folder is set to the mapped value. For example, if the specified selector 'Alpha' is a configuration that maps to the 'Gold' tag, then the persistent selector list for that folder is set to 'Gold'. Further, if the folder references a different vault (as identified by the REFERENCE keyword in the sync_project.txt file) and you are doing a recursive populate, the persistent selector list for any subfolder is also set to the mapped value.

Notes:
- The case where a ProjectSync configuration and its associated DesignSync tag have the same name is not considered configuration mapping; the persistent selector list is not modified by the populate operation.
- Only the populate command (not co, ci, and so on) resolves the selector you specify to a ProjectSync configuration, if one exists.
- DesignSync does not follow chained configuration maps. For example, if the same sync_project.txt file has a configuration A mapped to tag B and a configuration B mapped to tag C, DesignSync does not map A to C. Unexpected behavior can result. To avoid chained configuration maps, consider using separate naming conventions for configurations and tags.

- If an legacy module populate fails and the populate command was run with the -report brief option specified, you may not have enough information to determine where the failure occurred. If you rerun the populate with the -report brief mode, you will can locate the referenced object within the module hierarchy.

For information on how populate works on a legacy module or an href to a legacy module, see the description of -recursive option. See ProjectSync User's Guide for more information on ProjectSync projects

and configurations. See the "Working with Legacy Modules" book in DesignSync Data Manager User's Guide for more information about legacy modules.

Incremental Versus Full Populate (Legacy-based)

By default, the populate command attempts to perform an incremental populate which updates only those local objects whose corresponding vaults have changed. Avoiding a full populate improves the speed of the populate; however, some circumstances make a full populate necessary. In the following cases, DesignSync automatically performs a full populate:

- If you are populating with a different configuration to that of the work area (having used setselector, setvault, 'populate -version', or 'co -version' to change a selector), DesignSync performs a full populate. For example, if your last full populate specified the VendorA_Mem configuration, but you now want VendorB_Mem files, then DesignSync automatically performs a full (nonincremental) populate. If the selector you specify resolves to the same exact selector as that of the work area, DesignSync does perform the incremental populate. In this case, the selectors must be an exact match; for example, a selector which resolves to 'Main' does not match 'Main:Latest'. If you are populating with a new configuration, consider using the -force option to remove objects of the previous configuration from your work area.

- If you use the -lock option, DesignSync performs a full populate.

- If you use the -unifystate option, DesignSync performs a full populate.

- If you perform a nonrecursive populate on a subfolder, all of the folders above the subfolder are invalidated for subsequent incremental populate operations. Incremental populate works by exploiting the fact that if a folder is up-to-date, all of its subfolders are also up-to-date, making it unnecessary to recurse into them. Because a recursive populate was not performed for the subfolder, DesignSync cannot ensure that its subfolders are up-to-date; thus, all incremental populates are invalidated up the hierarchy.

- If you perform a nonrecursive populate on a folder, DesignSync essentially runs a full populate rather than the default incremental populate. Your next populate is incremental from the last recursive populate. If you have not previously run a recursive populate, the subsequent populate is a full populate.

- If a DesignSync REFERENCE resolves to a different selector than that of the work area from which the populate command is invoked,

DesignSync performs a full populate of the REFERENCED objects rather than an incremental populate. DesignSync compares the -version selector with the work area configuration rather than the mapped configuration, so do not use the -version selector to specify a mapped configuration. Instead, if you suspect the configuration map file has been updated, use the -version selector to remap the configuration by specifying the original selector. DesignSync then performs a full populate and follows the updated REFERENCEs.

- If the ProjectSync configuration file, sync_project.txt, has been updated through the ProjectSync interface (Project->Edit or Project->Configuration), thus updating the DesignSync REFERENCEs, DesignSync performs a full populate. If, however, the configuration in the sync_project.txt file is hand-edited and not updated using ProjectSync, you must specify the -full option to force a full populate.

Note: If you are using a mirror (by specifying -mirror or having a default fetch state of Links to mirror), an incremental populate does not necessarily fetch new objects checked in, nor remove links to objects deleted by team members until after the mirror has been updated.

For the following cases, perform a full populate instead of an incremental populate:

- If you have excluded a folder by using the -exclude or or -noemptydirs option with the populate command, a subsequent incremental populate will not necessarily process the folder of the previously excluded object. DesignSync does not automatically perform a full populate in this case. To guarantee that previously excluded objects are fetched, specify the -full option for the subsequent populate operation.

- If the ProjectSync configuration file, sync_project.txt, has been hand-edited, thus updating the legacy module REFERENCEs, use the -full option to perform a full populate. If, however, the sync_project.txt file has been changed through the ProjectSync interface (Project->Edit or Project->Configuration), DesignSync performs the full populate without your having to specify -full. For more information, see "Interaction with Legacy Modules" below.

- If the system clock on the SyncServer machine where your vault is located is turned back (for example, to correct clock skew between machines), you must perform a full (nonincremental) populate to synchronize the client and server metadata.

- If you interrupt a populate operation (using Control-c, for example), you should use populate -full on your next populate of that folder.

The default populate mode is -incremental; however, your project leader can set this default using SyncAdmin.

If you are updating mirrors, use the -incremental option. If you

specify the -full option, mirror updates can take a long time to complete.

Note: If you remove objects from the work area by using operating system commands rather than DesignSync commands, an incremental populate cannot fetch these objects. Perform a full populate or use the -unifystate or to fetch them.

Setting up Your Workspace (File-based)

Before you can use populate to maintain your workspace, you must set up your workspace.

Note: The Workspace Wizard from the DesignSync graphical user interface simplifies the task of setting up a work area by taking you through a step-by-step process.

The typical steps when you set up a new workspace are:

1. Associate a local folder with a vault folder. See the setvault command for details. This also creates the workspace root, if one does not already exist at the level of the local folder or above.

2. Optionally set the persistent selector list for the folder as part of the setvault command or with the setselector command. If you do not set the persistent selector list, it is inherited from the parent folder. This step is necessary only if you are working on a branch other than the default Trunk branch.

3. Optionally associate a local folder with a mirror directory. See the "setmirror" command for details. If the mirror directory for your project later changes, run the setmirror command from the same directory in which the original setmirror command was run. That will update the workspace's mirror association, which will be inherited by lower level directories. Run the populate command with the options '-recursive -mirror -unifystate' to correct existing workspace links to mirror files. This will correct the links so that they point to the mirror directory's new location.

4. Populate the work area with the specified design objects from the vault. Populate determines the set of versions from the persistent selector list or from the -version option, if applicable. Apply the -recursive option to create a local hierarchy that matches the vault's hierarchy. Without -recursive, populate only fetches the specified objects.

Incremental Versus Full Populate (File-based)

By default, the populate command attempts to perform an incremental

populate which updates only those local objects whose corresponding vaults have changed. Avoiding a full populate improves the speed of the populate; however, some circumstances make a full populate necessary. In the following cases, DesignSync automatically performs a full populate:

- If you are populating with a different configuration to that of the work area (having used setselector, setvault, 'populate -version', or 'co -version' to change a selector), DesignSync performs a full populate. For example, if your last full populate specified the VendorA_Mem configuration, but you now want VendorB_Mem files, then DesignSync automatically performs a full (nonincremental) populate. If the selector you specify resolves to the same exact selector as that of the work area, DesignSync does perform the incremental populate. In this case, the selectors must be an exact match; for example, a selector which resolves to 'Main' does not match 'Main:Latest'. If you are populating with a new configuration, consider using the -force option to remove objects of the previous configuration from your work area.

- If you use the -lock option, DesignSync performs a full populate.

- If you use the -unifystate option, DesignSync performs a full populate.

- If you perform a nonrecursive populate on a subfolder, all of the folders above the subfolder are invalidated for subsequent incremental populate operations. Incremental populate works by exploiting the fact that if a folder is up-to-date, all of its subfolders are also up-to-date, making it unnecessary to recurse into them. Because a recursive populate was not performed for the subfolder, DesignSync cannot ensure that its subfolders are up-to-date; thus, all incremental populates are invalidated up the hierarchy.

- If you perform a nonrecursive populate on a folder, DesignSync essentially runs a full populate rather than the default incremental populate. Your next populate is incremental from the last recursive populate. If you have not previously run a recursive populate, the subsequent populate is a full populate.

- If the ProjectSync configuration file, sync_project.txt, has been updated through the ProjectSync interface (Project->Edit or Project->Configuration), thus updating the DesignSync REFERENCEs, DesignSync performs a full populate. If, however, the configuration in the sync_project.txt file is hand-edited and not updated using ProjectSync, you must specify the -full option to force a full populate.

Note: If you are using a mirror (by specifying -mirror or having a default fetch state of Links to mirror), an incremental populate does not necessarily fetch new objects checked in, nor

remove links to objects deleted by team members until after the mirror has been updated.

For the following cases, perform a full populate instead of an incremental populate:

- If you have excluded a folder by using the -exclude, or -noemptydirs option with the populate command, a subsequent incremental populate will not necessarily process the folder of the previously excluded object. DesignSync does not automatically perform a full populate in this case. To guarantee that previously excluded objects are fetched, specify the -full option for the subsequent populate operation.

- For modules, DesignSync tracks changed members and therefore always performs an incremental populate. Specify a full populate to force data that has been manually removed, removed locally, or renamed locally to be fetched again from the server. If the file was renamed, you may have to specify the -force option as well.

  Also, specify a full populate if you have an unchanged, but out-of-date or out-of-sync version in your workspace to force DesignSync to fetch the up-to-date version of the object.

- If the ProjectSync configuration file, sync_project.txt, has been hand-edited, thus updating the legacy module REFERENCEs, use the -full option to perform a full populate. If, however, the sync_project.txt file has been changed through the ProjectSync interface (Project->Edit or Project->Configuration), DesignSync performs the full populate without your having to specify -full. For more information, see "Interaction with Legacy Modules" below.

- If the system clock on the SyncServer machine where your vault is located is turned back (for example, to correct clock skew between machines), you must perform a full (nonincremental) populate to synchronize the client and server metadata.

- If you interrupt a populate operation (using Control-c, for example), you should use populate -full on your next populate of that folder.

The default populate mode is -incremental; however, your project leader can set this default using SyncAdmin.

If you are updating mirrors, use the -incremental option. If you specify the -full option, mirror updates can take a long time to complete.

Note: If you remove objects from the work area by using operating system commands rather than DesignSync commands, an incremental populate cannot fetch these objects. Use the -unifystate or -full option to fetch them.

How Populate Handles Retired Objects (File-based)

When you populate with the Latest versions of design objects from a given branch, DesignSync does not populate objects for which that branch is retired. Objects in your local work area whose branches have been retired from the vault are not deleted during the populate operation unless you specify -force.

It is important to note that objects on retired branches remain part of past configurations. When you use the populate command to retrieve a configuration other than 'Latest', objects from retired branches are fetched. The populate command fetches objects from retired branches, thereby preserving past configurations, if the selector used for the operation is any of the following:

- A version tag other than 'Latest', even if the version tag points to the Latest version
- A version number, even if that number corresponds to the Latest version
- <branchtag>:Date(<date>) or <branchtag>:VaultDate(<date>)

Note: If the selector specifies a branch in the form '<branchtag>:', DesignSync augments the selector to be <branchtag>:Latest, meaning, 'Get the Latest version from the specified branch'. In this case, objects from retired branches are not fetched.

Note: For information about how retired files by cross-branch merge operations, see "Merging Across Branches."

Merging Across Branches (File-based)

In multi-branch environments, you use the populate command to merge branches. In many cases, a new branch that is created is eventually merged back into the main development branch.

The branch being merged is populated into a workspace containing the destination branch using the populate command with the -merge and -overlay options. This type of merge is called "cross-branch merge."

As with all populate operations, cross-branch merging uses the filter and exclude filter lists set on the workspace, in the command defaults system,on the command line.

Merging includes two basic types of merging: file contents, and structural changes.

- File content merging:
  File content merging is applicable to all DesignSync objects. DesignSync merges the contents of files with the same natural path to the best of its ability. If the files are binary files which cannot be merged, populate returns an error message.

- Structural changes for DesignSync objects.
  Structural changes for DesignSync objects are non-content based changes to the DesignSync objects that can affect the merge

results.

- Removed objects: If an object is present in the local workspace, but not in the merge version, the object in the local workspace is unchanged. If you want to remove it from the merged version, you must explicitly remove or retire the object.

- Added objects: If an object is not present in the local workspace, but is present in the merge version, the object is added to the local workspace. The merge action sets the following local metadata properties:

  - The current version is set to the fetched version, providing a meaningful branch-point version when you check the object into branch A.
  - The current branch information is undefined.
  - The persistent selector list for the object may be augmented to ensure that branch A is automatically created when you check in the object, thus eliminating the need to use ci -new. The following list explains how the persistent selector list is handled by the operation.
    1. If the first selector in the persistent selector list is a VaultDate() or Auto() selector, then the persistent selector list is not modified.
    2. If the first selector is of the form <branch>:<version>, then the first selector is modified to be Auto(<branch>).
    3. Otherwise, the first selector is modified to be Auto(<selector>). The object may be automatically checked in to the DesignSync vault, depending on the value of the persistent selector.

- Retired objects:

  - If the object is active in the workspace and retired on the branch version, the workspace version is unchanged.
  - If the object is retired or does not exist in the workspace, and is retired or does not exist on the branch, the workspace version is unchanged.
  - If the object is retired in the workspace and active on the branch version, the version from the branch version is merged with the workspace version. The object remains retired and must be unretired in order to be checked in.

After a cross-branch merge has been performed, you can view the status of the affected files using the ls command with the -merged <state> -report D options. The -merged option allows you to restrict the list to a particular type of merge operation (add, remove, rename, all) and the -report D option shows you the current state of the object in your workspace. For more information, see the ls command help.

When a merge is performed on a DesignSync object, a merge edge is created automatically to maintain a list of the changes incorporated by the merge. This identifies a closer-common ancestor to provide for quicker subsequent merges.

For more information about merging, see the -merge and -overlay options, and the DesignSync Data Manager User's Guide topic: "What Is Merging?"

Populate Versus Checkout (File-based)

The co and populate commands are similar in that they retrieve versions of objects from their vaults and place them in your work area. They differ in several ways, most notably:

- You typically use the co command to operate on objects that you already have locally, whereas populate updates your work area to reflect the status of the vault.
- The co command considers the persistent selector list for each object that is checked out, whereas populate only considers the persistent selector list for the folder that is being populated.

Note: The co and populate commands are gradually being merged.

Understanding the Output (File-based)

The populate command provides the option to specify the level of information the command outputs during processing. The -report option allows you to specify what type of information is displayed:

If you run the command with the -report brief option, the populate command outputs a small amount of status information including, but not limited to:

- Failure messages.
- Warning messages.
- Informational messages concerning the status of the populate
- Success/failure/skip status

If you do not specify a value, or the command with the -normal option, the populate command outputs all the information presented with -report brief and the following additional information:

- Informational messages for objects that are updated by the populate operation.
- Messages for objects excluded from the operation (due to exclusion filters or explicit exclusions).

If you run the command with the -report verbose option, the populate command outputs all the information presented with -report normal and the following additional information:

- Informational message for every object examined but not updated.

If you run the command with the -report error option, the populate command outputs the following information:

- Failure messages.
- Warning messages.
- Success/failure/skip status messages.

Forcing, Replacing, and Non-Replacing Modes (File-based)

You can use these three modes to specify how the populate command updates your work area:

- Forcing mode (specified with the -force option) synchronizes your work area with the incoming data, including locally modified objects. In this mode, the populate command updates the managed objects in your work area. It replaces or removes managed objects regardless of whether the objects have been locally modified. Thus, forcing modifies your work area to match the set of objects being fetched. Note: The default -noforce option operates as if -replace has been specified.

- Replacing mode (specified with the -replace option) also synchronizes your work area with the incoming data, but without affecting locally modified objects (the default behavior).

  Note: Retired files that have been kept or re-added to the workspace are considered locally modified.

  Replacing mode, unlike forcing mode, leaves intact managed objects that have been locally modified.

- Non-replacing mode (specified with the -noreplace option) is the least disruptive mode; this mode might require you to clean up the resulting work area data.

  In this mode, the populate command takes the incoming data and overlays it on top of the existing work area's data. It leaves intact both managed objects with local modifications and managed objects that are not members of the module being fetched. Thus, the work area essentially becomes a union of the data from the previous version and that of the module being fetched.

  Non-replacing mode, unlike forcing mode, leaves intact any objects that have been locally modified, and, unlike the replacing mode, leaves unmodified managed objects intact. See the -[no]replace option below for more details.

Notes:
- Unmanaged objects in your work area are not affected by any of these modes.
- The following are illegal combinations of options: -replace and -noforce, as well as inverse options, such as -replace and -noreplace.

SYNOPSIS

```
populate [-[no]connectinstances] [-[no]emptydirs]
         [-exclude <object>[,<object>...]] [-filter <string>]
```

```
[-[no]force] [-full | -incremental] [-hreffilter <string>]
[-hrefmode {static | dynamic | normal}]
[[-lock [-keys <mode> | -from {local | vault}]] |
[-get [-keys <mode> | -from {local | vault}]]
[-share] | [-mirror] | [-reference] [-lock -reference] ]
[-log <filename>]  [-mcachemode <mcache_mode>]
[-mcachepaths <path_list>] [-[no]merge]
[-modulecontext <context>] [-[no]new]]
[[-overlay <selector>[,<selector>...]]|
[-version <selector>[,<selector>...]]] [-path <path>]
[-[no]recursive] [-[no]replace]
[-report {error|brief|normal|verbose}] [-[no]retain]
[-savelocal <value>] [-target <module_configuration_url>]
[-trigarg <arg>] [-[no]unifystate] [-view view1[,view2,...]]
[-xtras <list>] [--] [<argument> [<argument>...]]
```

ARGUMENTS

- Server Module URL (Module-based)
- Workspace Module (Module-based)
- Module Folder (Module-based)
- Module Member (Module-based)
- Hierarchical Reference (Module-based)
- External Module (Module-based)
- DesignSync Object (File-based)
- DesignSync Folder (File-based)

```
The populate command accepts multiple arguments. If you want
to populate the current folder, you need not specify an
argument.  Otherwise, specify one or more of the following
arguments:
```

Server Module URL (Module-based)

```
<server module>     Fetches the specified module from its vault.
                    For an initial populate of a module, you must
                    specify the module's server URL in the format:
                    sync://<machine>:<port>/Modules/<category>/
                    <module_name>[;<selector>].

                    The populate fetches all objects within the
                    module, but does not follow hierarchical
                    references (hrefs) by default; specify the
                    -recursive option to follow hrefs and thus
                    fetch the module's submodules.
```

Workspace Module (Module-based)

<workspace module>	Fetches the specified module from its vault, or updates the module to the appropriate module version specified by the selector in use. The populate fetches all objects within the module, but does not follow hierarchical references (hrefs) by default; specify the -recursive option to follow hrefs and thus fetch the module's submodules.

Module Folder (Module-based)

<module folder>	Populates objects in the specified folder regardless of which module the files belong to. Specify the -recursive option to recurse within the specified folder. Populate in this case, does not follow hierarchical references (hrefs). Note: To populate a module folder, the folder must already exist in the workspace. If you specify the -modulecontext option, the populate command updates the items belonging to the specified module in the specified folder and all the sub-folders. If you use the -recursive option in addition to the -modulecontext option, populate fetches any items from relevant sub-modules that fall within the folder specified (or its sub-folders.) Specify the module folder as an absolute path or a relative path. If you specify a relative path, it is assumed to be relative to the current directory or that specified by the -path option. Note: In previous releases, if the directory that was being populated was part of a legacy module, the entire module and not just the module members in the directory got populated.

Module Member (Module-based)

<module member>	Fetches the module member. You can specify the -modulecontext option if more than one module exists in the workspace. Note: The -modulecontext option is not normally needed, as the system knows what module each member belongs to. When there are

multiple overlapping modules and you are fetching an object that is not currently in the workspace (for example, to fetch something that was originally filtered, or was removed with rmfile), the -modulecontext option can be used to identify the module from which the object should be fetched.

You can also provide the version-extended name if necessary. A version-extended name is a filename followed by a semicolon and a version number or tag name (for example, top.v;1.2 or top.v;rel13). In this case, DesignSync fetches the specific version of the member vault instead of fetching the version of this object that belongs with the module version.
Note: If you specify the version-extended name, populate ignores the -version option.

Hierarchical Reference (Module-based)

<href> Fetches the referenced target (submodule) identified by the hierarchical reference (href). You can use -hreffilter to exclude submodules. To include submodules, enter the href as the argument of the populate command. To indicate the module context of the href, use the -modulecontext option.

Note: You can only specify hrefs directly within the specified module. For example, if a module Chip has an href to module CPU, and module CPU has an href to module ALU, you cannot reference the ALU. Thus, the following command invocations are invalid: 'populate -modulecontext Chip ALU' and 'populate -modulecontext Chip CPU/ALU'.

External Module (Module-based)

<external module> Specifies the module instance of the external module in the workspace or the URL of the external module version to which you wish to create the connection. An external module is an object or set of objects managed by a different code management system but available for viewing and integration through DesignSync.
Specify the external module in the workspace as a module instance name.
Specify the external module Server URL as follows:

```
sync[s]:///ExternalModule/<external-type>/<external-data>
       where ExternalModule is a constant that identifies
       this URL as an external module URL,
       <external-type> is the name of the external module
       procedure, and <external-data> contains the
       parameters and options to pass to the
       procedure. These parameters and options can be
       passed from the procedure to the external code
       management system or to DesignSync.

       Note: In order to specify an external module, you
      must have previously populated the module in the
      workspace. You may also specify the external
      module by href name only.
```

DesignSync Object (File-based)

```
<DesignSync object> Fetches the object from its vault.
```

DesignSync Folder (File-based)

```
<DesignSync folder> Fetches the contents of the specified folder.
                    You can also use the -path option to specify
                    a folder to be fetched.
```

OPTIONS

- -[no]connectinstances (Module-based)
- -[no]emptydirs
- -exclude (Module-based)
- -exclude (File-based)
- -filter (Module-based)
- -[no]force (Module-based)
- -[no]force (File-based)
- -from
- -full
- -get (Module-based)
- -get (File-based)
- -hreffilter (Module-based)
- -hrefmode (Module-based)
- -incremental
- -keys (Module-based)
- -keys (File-based)
- -lock (Module-based)
- -lock (Legacy-based)
- -lock (File-based)

-[no]connectinstances (Module-based)

```
-[no]connectinstances   This option determines how to handle updating
                        hierarchical reference within a top-level
                       module.

                        If your workspace is set up
                        in a peer structure, containing your
                        top-level module and modules which are
                        referenced submodules, but have been
                        populated independently, then when
                        your workspace is populated non-recursively,
                        DesignSync does not recognize the connection
                        between the modules.  When populated
                        recursively, DesignSync may change the
```

selector of the submodules to match the hierarchical reference definition. The -connectinstances option allows you to populate the top-level module, recognizes that the peer modules are, in fact, referenced submodules, and creates the relationship accordingly, but does not update the selector to match the hierarchical reference definition.

This option is mutually exclusive with -recursive which updates the href to the referenced peer module.

The -noconnectinstances option does not establish or identify a hierarchical relationship with referenced peer modules. (Default)

Notes:
* You can use the -connectinstances option with the -hreffilter option to identify specific submodules instead of updating the relationships for the entire module hierarchy.

* The submodule must match the target module and relative path specified in the hierarchical reference in order to the update the href.

-[no]emptydirs

-[no]emptydirs	Determines whether empty directories are removed or retained when populating a directory. Specify -noemptydirs to remove empty directories or -emptydirs to retain them. The default for the populate operation is -noemptydirs. For example, if you are creating a directory structure to use as a template at the start of a project, you may want your team to populate the empty directories to retain the directory structure. In this case, you would specify 'populate -rec -emptydirs'. If a populate operation using -noemptydirs empties a directory of its objects and if that directory is part of a managed data structure (its objects are under revision control), then the populate operation removes the empty directory. If the empty directory is not part of a managed data structure, then the

operation does not remove the directory or its subdirectories. (A directory is considered part of the managed data structure if it has a corresponding folder in the DesignSync vault or if it contains a .SYNC client metadata directory.)

Notes:
- When used with 'populate -force -recursive', the -noemptydirs option removes empty directories that have never been managed.
- When used with the -mirror option, the -noemptydirs option does not remove empty directories (unless -force -recursive is used) and does not populate directories that are empty in the mirror.
- When the -noemptydirs option is used with '-report verbose', the command might output messages that additional directories are being deleted. Those are directories created by the populate, to mimic the directory structure in the vault. If no data is fetched into those directories (because no file versions match the selector), then those empty directories are deleted.

If you do not specify -emptydirs or -noemptydirs, the populate command follows the DesignSync registry setting for "Populate empty directories". By default, this setting is not enabled; therefore, the populate operation removes empty directories. To change the default setting, your Synchronicity administrator can use the SyncAdmin tool. For information, see SyncAdmin help. You typically want consistent behavior for all users, so adding the setting to the site registry is recommended.

-exclude (Module-based)

`-exclude <objects>` Specifies a comma-separated list of objects (files, collections, folders, or module objects) to be excluded from the operation. Wildcards are allowed.

Note: Use the -filter option to filter module objects. You can use the -exclude option, but the -filter option lets you include and exclude module objects. If you use both the -filter and -exclude options, the strings specified using -exclude take precedence.

If you exclude objects during a populate, a subsequent incremental populate will not necessarily process the folders of the previously excluded objects. DesignSync does not automatically perform a full populate in this case. To guarantee that previously excluded objects are fetched, specify the -full option for the subsequent populate operation.

The '-exclude' option is ignored if it is included in a 'populate -mirror' operation.

Do not specify paths in your arguments to -exclude. Before operating on each object (such as during a recursive populate), DesignSync compares the object's leaf name (with the path stripped off) to the exclude list to see if there is a match. Because the object's path is removed, the object will not match any object in the exclude list specified with a path. For example, if you specify '-exclude bin/*.exe', you will not successfully exclude bin/foo.exe or any other *.exe file. You need to instead specify '-exclude *.exe', or '-exclude foo.exe' if you want to exclude only 'foo.exe'. The result is that you cannot exclude a specific instance of an object -- you exclude all matching objects.

In addition to objects you specify using the -exclude option, the field, "These objects are always excluded", from the Tools->Options-> General->Exclude Lists dialog box in DesignSync lists objects that are always excluded from revision-control operations.

Note: Do not exclude members when you are fetching a module into the module cache; users cannot link to or copy from a filtered module in a module cache.

-exclude (File-based)

-exclude <objects> Specifies a comma-separated list of objects (files, collections, or folders) to be excluded from the operation. Wildcards are allowed.

If you exclude objects during a populate, a subsequent incremental populate will not necessarily process the folders of the previously excluded objects. DesignSync does not automatically perform a full populate in this case. To guarantee that previously excluded objects are fetched, specify the

```
                    -full option for the subsequent populate
                    operation.

                    The '-exclude' option is ignored if it is
                    included in a 'populate -mirror' operation.

                    Do not specify paths in your arguments to
                    -exclude. Before operating on each object (such
                    as during a recursive populate), DesignSync
                    compares the object's leaf name (with the path
                    stripped off) to the exclude list to see if there
                    is a match. Because the object's path is removed,
                    the object will not match any object in the
                    exclude list specified with a path. For example,
                    if you specify '-exclude bin/*.exe', you will not
                    successfully exclude bin/foo.exe or any other
                    *.exe file. You need to instead specify '-exclude
                    *.exe', or '-exclude foo.exe' if you want to
                    exclude only 'foo.exe'. The result is that you
                    cannot exclude a specific instance of an object
                    -- you exclude all matching objects.

                    In addition to objects you specify using the
                    -exclude option, the field, "These objects are
                    always excluded", from the Tools->Options->
                    General->Exclude Lists dialog box in
                    DesignSync lists objects that are always
                    excluded from revision-control operations.
```

-filter (Module-based)

```
 -filter <string>   Specify one or more extended glob-style
                    expressions to identify an exact subset of module
                    objects on which to operate. Use the -exclude
                    option to filter out DesignSync objects that are
                    not module objects.

                    The -filter option takes a list of expressions
                    separated by commas, for example:
                      -filter +top*/.../*.v,-.../a*

                    Prepend a '-' character to a glob-style expression
                    to identify objects to be excluded (the default).
                    Prepend a '+' character to a glob-style expression
                    to identify objects to be included.  Note that if
                    the list of expressions begins with an include
                    character ('+'), the filter excludes all objects
                    except those that match the include string.

                    Specify the paths in your glob-style expressions
                    relative to the current directory, because
                    DesignSync matches your expressions relative to
                    that directory. For submodules followed through
                    hrefs, DesignSync matches your expressions against
```

the objects' natural paths -- their full relative paths. For example, if a module, Chip, references a submodule, CPU, and CPU contains a file, '/libs/cpu/cdsinfo.tag', DesignSync matches against '/libs/cpu/cdsinfo.tag', rather than matching directly within the 'cpu' directory.

If your design contains symbolic links that are under revision control, DesignSync matches against the source path of the link rather than the dereferenced path. For example, if a symbolic link exists from 'tmp.txt' to 'tmp2.txt', DesignSync matches against 'tmp.txt'. Similarly for hierarchical operations, DesignSync matches against the unresolved path. If, for example, a symbolic link exists from dirA to dirB and dirB contains 'tmp.txt', DesignSync matches against 'dirA/tmp.txt'.

The extended glob-style expressions you use to filter the objects are standard glob-style expressions, but they are extended so that you can use the "..." syntax to indicate that the expression matches any number of directory levels. For example, the expression, "top/.../lib/*.v" matches *.v files in a directory path that begins with "top", followed by zero or more levels, with one of those levels containing a lib directory. The command traverses the directory structure. If a directory name matches an exclude clause of the filter, then the entire directory and all its contents are filtered (the command stops descending at that point), otherwise the command continues traversing the directory structure searching for matching objects.

If this is the first time a module is being populated, the filter becomes a persistent filter for the module, just as if you had invoked the setfilter command. For subsequent operations on the module, DesignSync applies persistent filters first, followed by those set using the -filter, -hreffilter, and -exclude options.

Note: If a populate specifies a -filter value to filter out objects that were previously populated, the populate is not considered complete. In this case, the workspace module does not match the module in the vault; thus, the module version is not updated. Also, a subsequent incremental populate will not necessarily process the folders of the previously excluded objects. DesignSync does not automatically perform a full populate in this case. To guarantee that previously excluded objects are fetched, specify the

-full option for the subsequent populate operation.

Although the -filter option takes precedence over persistent filters, it does not override the exclude list set using SyncAdmin's General=>Exclude Lists tab; the items in the exclude list are combined with the filter expression. For example, an exclude list of "*%,*.reg" combined with '-filter .../*.doc' is equivalent to:
'-filter .../*.doc,.../*%,.../*.reg'.

Note: Do not filter a module that you are fetching into the module cache; users cannot link to or copy from a filtered module in a module cache.

-[no]force (Module-based)

-[no]force

Specifies whether to overwrite locally modified objects in order to match the workspace to the data being requested. To do so, the populate operation deletes locally managed objects that are not part of the populate command line, deleting objects that have been filtered out. 'populate -force' only removes managed data, not unmanaged data. For module objects, the -force option removes objects from modules if they have been added by the add command, but have never been checked in. Again, although DesignSync removes these objects from the module manifest, it does not remove the unmanaged data. Also, if you specify -force while populating a module that overlaps with another module, the -force option does not remove data from the other module.

Use this option with caution, because you might not be able to retrieve lost changes.

By default (-noforce):
- Locally modified objects are not overwritten by the populate operation. Specify -force if you want to overwrite locally modified objects. If the object is locked, the object is unaffected by the populate operation whether -force is specified or not.
- Objects that are not part of the specified module remain in your work area. If you want to delete objects that are not part of the configuration, specify -force. Unmanaged objects are never deleted.

Using -force with -unifystate changes the state

of all objects including locally modified objects, in which case, local modifications are overwritten and objects are fetched according to the specified state or the default fetch state.

Using -force with -noemptydirs for populate removes all existing empty directories from the workspace unless the directories themselves are members of the module.

The -force option is mutually exclusive with both the -overlay and -noreplace options.

-[no]force (File-based)

-[no]force

Specifies whether to overwrite locally modified objects in order to match the workspace to the data being requested. To do so, the populate operation deletes locally managed objects that are not part of the populate command line, deleting objects that have been filtered out. 'populate -force' only removes managed data, not unmanaged data.

Use this option with caution, because you might not be able to retrieve lost changes.

By default (-noforce):

- Locally modified objects are not overwritten by the populate operation. Specify -force if you want to overwrite locally modified objects. If the object is locked, the object is unaffected by the populate operation whether -force is specified or not.
- Objects that are not part of the specified configuration remain in your work area. If you want to delete objects that are not part of the configuration, including retired objects, specify -force. Unmanaged objects are never deleted.

The behavior of 'populate -mirror' without -force is different from populate with other states (see the description of -mirror). Therefore, -force with -mirror has the additional effect of changing the state of existing objects in your work area, resulting in a hierarchy that exactly reflects the mirror directory.

Using -force with -unifystate changes the state of all objects including locally modified objects, in which case, local modifications are overwritten and objects are fetched according to

the specified state or the default fetch state.

Using -force with -noemptydirs for populate removes all existing empty directories from the workspace.

The -force option is mutually exclusive with both the -overlay and -noreplace options.

-from

-from <where>

Specifies whether the object is fetched from the vault ('-from vault') or from the cache or mirror ('-from local'). By default, DesignSync fetches from the cache or mirror ('-from local'), a performance optimization specific to the 'co -lock', 'co -get', 'populate -lock', and 'populate -get' commands. For details, see the Performance Optimization Overview in the DesignSync Data Manager Administrator's Guide. Note that this option is silently ignored when the optimization is not possible, including when the -keys option is specified.

The -from option can only be used with the -lock or -get fetch modes. It cannot be used with the -share, -mirror, -reference, or the -lock -reference combination fetch modes. If the -keys option is specified with the -from option, the -from option is silently ignored.

-full

-full

Performs a non-incremental populate by processing all objects and folders.

Note: DesignSync performs an incremental populate by default. It automatically reverts to a full populate when necessary. For more information, see the "Incremental Versus Full Populate" section in the description.

To change the default populate mode, your Synchronicity administrator can use the SyncAdmin tool.

Note: Do not use the -full option to change the states of objects in your work area (for example, changing from locked to unlocked objects or unlocked objects to links to

```
                    the cache). DesignSync changes the states
                    of only those objects that need an
                    update. Use the -unifystate option to
                    change the state of objects in your work
                    area.
```

-get (Module-based)

```
 -get               Fetch unlocked copies.

                    You can change whether the local object is
                    read-only (typical when using the locking
                    model) or read/write (typical when using the
                    merging model) by default by using the
                    "Check out read only when not locking" option
                    from the Tools->Options->General dialog box in
                    the DesignSync graphical interface. Your project
                    leader can also set this option site-wide
                    using SyncAdmin.

                    This option is the default object-state option
                    unless a default fetch preference has been
                    defined. See the "fetch preference" help topic
                    for more information.

                    Using -force with -noemptydirs for
                    'populate -get' removes all existing empty
                    directories. Using -force with -emptydirs,
                    however, creates empty directories for
                    corresponding empty vault folders. Note that
                    the populate command ignores the
                    -noemptydirs option when operating on
                    modules, because folders are members of
                    their corresponding modules and therefore
                    cannot be removed.

                    The -get option is mutually exclusive with the
                    other fetch modes: -lock, -share, -mirror, and
                    -reference.

                    Note: To replace mcache links with physical
                    copies of module members, use the -mcachemode
                    server option,
```

-get (File-based)

```
 -get               Fetch unlocked copies.

                    You can change whether the local object is
                    read-only (typical when using the locking
                    model) or read/write (typical when using the
                    merging model) by default by using the
                    "Check out read only when not locking" option
```

from the Tools->Options->General dialog box in the DesignSync graphical interface. Your project leader can also set this option site-wide using SyncAdmin.

This option is the default object-state option unless a default fetch preference has been defined. See the "fetch preference" help topic for more information.

Using -force with -noemptydirs for 'populate -get' removes all existing empty directories. Using -force with -emptydirs, however, creates empty directories for corresponding empty vault folders.

The -get option is mutually exclusive with the other fetch modes: -lock, -share, -mirror, and -reference.

-hreffilter (Module-based)

-hreffilter <string>	Excludes href values during recursive operations on module hierarchies. Because hrefs link to submodules, you use -hreffilter to exclude particular submodules. Note that unlike the -filter option which lets you include and exclude items, the -hreffilter option only excludes hrefs and, thus, their corresponding submodules. Note: When populating a workspace with symbolic links to a module cache, the -hreffilter option does not apply and is silently ignored. Specify the -hreffilter string as a glob-style expression. The href filter can be specified either as a simple href filter or as a hierarchical href filter. A simple href filter is a simple leaf module name; you cannot specify a path. DesignSync matches the specified href filter against hrefs anywhere in the hierarchy. Thus, DesignSync excludes all hrefs of this leaf name; you cannot exclude a unique instance of the href. A hierarchical href filter specifies a path and a leaf submodule, for example JRE/BIN excludes the BIN submodule only if it is directly beneath JRE in the hierarchy. When creating a hierarchical href filter, you do not specify the top-level module of the

hierarchy. If you want to filter using the top-level module, you begin the hreffilter with /, for example, "/JRE," would filter any JRE href referenced by the top-level module.

Note: You can use wildcards with both types of hreffilter, however, if a wildcard is used as the lone character in hierarchical href, it only matches a single level, for example: "JRE/*/BIN" would match a hierarchy like "JRE/SUB/BIN" but would not match "JRE/BIN" or "JRE/SUB/SUB2/BIN".

You can prepend the '-' exclude character to your string, but it is not required. Because the -hreffilter option only supports excluding hrefs, a '+' character is interpreted as part of the glob expression.

If this is the first time a module is being populated, the filter becomes a persistent filter for the module, just as if you had invoked the setfilter command. For subsequent operations on the module, DesignSync applies persistent filters first, followed by those set using the -filter, -hreffilter, and -exclude options.

Note: Hierarchical hreffilters can only be specified during an initial populate. To add, change, or remove a hierarchical hreffilter after the initial populate, you must use the setfilter command.

Whereas the -filter option can prevent a populate from being complete, thus preventing the version from being updated, the -hreffilter option does not prevent the version from being updated. The -hreffilter option prevents particular submodules from being fetched, but the failure to fetch a submodule does not affect the updating to a new version.

Note: Do not filter a module that you are fetching into the module cache; users cannot link to or copy from a filtered module in a module cache.

-hrefmode (Module-based)

-hrefmode

For a recursive populate, determines whether to populate statically-specified submodules or dynamically-evaluated submodules.

Valid values are:
- dynamic - Expands hrefs at the time of the populate operation to identify the version

of the submodules to be populated.
- static - Populates with the submodules versions referenced by the hrefs when the module version was initially created.
- normal - Expands the hrefs at the time of the populate operation until it reaches a static selector. If the reference uses a static version, the hrefmode is set to 'static' for the next level of submodules to be populated; otherwise, the hrefmode remains 'normal' for the next level. (Default). This behavior can be changed using the "HrefModeChangeWithTopStaticSelector" registry key to determine how hrefs are followed.

Notes:
- If the -hrefmode option is used, it is stored for subsequent populates; You do not have to specify the href mode again unless a different mode is required.
- Use of the -hrefmode option is mutually exclusive with use of the -lock option.
- If an href is created with a mutable version tag, and that version tag has moved, you must use dynamic mode (-hrefmode dynamic) to populate your workspace with the new tagged version. If you want the workspace to continue to point to the original version, you should populate with normal or static mode.
- If you are fetching modules into the module cache, use the static mode (-herfmode static). You can only link to statically fetched module versions. See DesignSync Data Manager Administrator's Guide: "Setting up a Module Cache" for more information.

-incremental

-incremental

Performs a fast populate operation by updating only those folders whose corresponding vault folders contain modified objects.
Note: DesignSync performs an incremental populate by default. It automatically reverts to a full populate when necessary.
For more information, see the "Incremental Versus Full Populate" section in the description.

To change the default populate mode, your Synchronicity administrator can use the SyncAdmin tool.

Note: Do not use the -incremental option to change the states of objects in your work area (for example, changing from locked to unlocked objects or unlocked objects to links to the cache). DesignSync changes the states of updated objects only. For an incremental populate, DesignSync only processes folders that contain objects that need an update. State changes, therefore are not guaranteed. Use the -unifystate option to change the state of objects in your work area.

-keys (Module-based)

-keys <mode>

Controls processing of vault revision-control keywords in populated objects. Note that keyword expansion is not the same as keyword update. For example, the $Date$ keyword is updated only during checkin; its value is not updated during checkout or populate. The -keys option only works with the -get and -lock options. If you use the -share or -mirror option, keywords are automatically expanded in cached or mirrored objects, as if the '-keys kkv' option was used.

Available modes are:

kkv - (keep keywords and values) The local object contains both revision control keywords and their expanded values; for example, $Revision: 1.4 $.

kk - (keep keywords) The local object contains revision control keywords, but no values; for example, $Revision$. This option is useful if you want to ignore differences in keyword expansion, such as when comparing two different versions of an object.

kv - (keep values) The local object contains expanded keyword values, but not the keywords themselves; for example, 1.4. This option is not recommended if you plan to check in your local objects. If you edit and then check in the objects, future keyword substitution is impossible, because the value without the keyword is interpreted as regular text.

ko - (keep output) The local object contains the same keywords and values as were present at check in.

The -keys option can only be used with the -lock or -get fetch modes. It cannot be used with the -share, -mirror, -reference, or the -lock -reference combination fetch modes. If the -keys option is specified with the -from option, the -from option is silently ignored.

Note: When a module member is checked out with a lock, the locker keyword is not updated for the lock operation and remains null.

-keys (File-based)

-keys <mode>

Controls processing of vault revision-control keywords in populated objects. Note that keyword expansion is not the same as keyword update. For example, the $Date$ keyword is updated only during checkin; its value is not updated during checkout or populate. The -keys option only works with the -get and -lock options. If you use the -share or -mirror option, keywords are automatically expanded in cached or mirrored objects, as if the '-keys kkv' option was used.

Available modes are:

kkv - (keep keywords and values) The local object contains both revision control keywords and their expanded values; for example, $Revision: 1.4 $.

kk - (keep keywords) The local object contains revision control keywords, but no values; for example, $Revision$. This option is useful if you want to ignore differences in keyword expansion, such as when comparing two different versions of an object.

kv - (keep values) The local object contains expanded keyword values, but not the keywords themselves; for example, 1.4. This option is not recommended if you plan to check in your local objects. If you edit and then check in the objects, future keyword substitution is impossible, because the value without the keyword is interpreted as regular text.

ko - (keep output) The local object contains the same keywords and values as were present at check in.

The -keys option can only be used with the -lock or -get fetch modes. It cannot be used with the

-share, -mirror, -reference, or the -lock -reference combination fetch modes. If the -keys option is specified with the -from option, the -from option is silently ignored.

-lock (Module-based)

-lock

Lock the branch of the specified version for each module member object that is populated. Only the user who has the lock can check in a newer version of the object on that branch.

The -lock option does not lock not the module branch. In so doing, the -lock option makes the members writable in the workspace, and converts cached objects to full copies. To lock the module branch itself without making members writable, use the lock command.

Use the -lock option with the -reference option to populate with locked references. For more information, see the -lock -reference option. Locked references are useful if you intend to generate objects and want to lock them before regenerating, as opposed to editing the previous version. Upon generation of the objects, they automatically become locked copies rather than locked references. Obtaining locked references for generated objects is faster because DesignSync does not fetch the previously generated objects. If the objects exist already in the workspace, DesignSync deletes them. If the objects exist and are locally modified, the operation fails. If you intend to overwrite the modifications, use -force to create the locked references. If the default fetch state is 'reference' and you specify the -lock option without the -reference option, DesignSync leaves locked copies of the objects in your workspace; you must explicitly apply the -reference option with the -lock option if you want locked references in your workspace.

The -lock option is mutually exclusive with the fetch modes: -get, -share, and -mirror and mutually exclusive with -recursive. The -lock option can be used with the -merge option.

Notes:

- If you specify 'populate -lock', then by default the populate operation also uses the '-from local' option. The result is that the populate operation locks the object in the

vault and keeps local modifications in your workspace. See the -from option for information.

o When a module member is checked out with a lock, the locker keyword is not expanded with the locker name.

-lock (Legacy-based)

-lock

Lock the branch of the specified version for each object that is populated. Only the user who has the lock can check in a newer version of the object on that branch.

Use the -lock option with the -reference option to populate with locked references. For more information, see the -lock -reference option. Locked references are useful if you intend to generate objects and want to lock them before regenerating, as opposed to editing the previous version. Upon generation of the objects, they automatically become locked copies rather than locked references. Obtaining locked references for generated objects is faster because DesignSync does not fetch the previously generated objects. If the objects exist already in the workspace, DesignSync deletes them. If the objects exist and are locally modified, the operation fails. If you intend to overwrite the modifications, use -force to create the locked references. If the default fetch state is 'reference' and you specify the -lock option without the -reference option, DesignSync leaves locked copies of the objects in your workspace; you must explicitly apply the -reference option with the -lock option if you want locked references in your workspace.

The -lock option is mutually exclusive with the fetch modes: -get, -share, and -mirror, and with -merge option.

Notes:

o If you specify 'populate -lock', then by default the populate operation also uses the '-from local' option. The result is that the populate operation locks the object in the vault and keeps local modifications in your workspace. See the -from option for information.

o If you use 'populate -lock -recursive' to fetch or update a module configuration

hierarchy, populate locks only the objects associated with the upper-level module (the module configuration specified as the target of the command).

-lock (File-based)

-lock

Lock the branch of the specified version for each object that is populated. Only the user who has the lock can check in a newer version of the object on that branch.

Use the -lock option with the -reference option to populate with locked references. For more information, see the -lock -reference option. Locked references are useful if you intend to generate objects and want to lock them before regenerating, as opposed to editing the previous version. Upon generation of the objects, they automatically become locked copies rather than locked references. Obtaining locked references for generated objects is faster because DesignSync does not fetch the previously generated objects. If the objects exist already in the workspace, DesignSync deletes them. If the objects exist and are locally modified, the operation fails. If you intend to overwrite the modifications, use -force to create the locked references. If the default fetch state is 'reference' and you specify the -lock option without the -reference option, DesignSync leaves locked copies of the objects in your workspace; you must explicitly apply the -reference option with the -lock option if you want locked references in your workspace.

The -lock option is mutually exclusive with the fetch modes: -get, -share, and -mirror and with the -merge option.

Notes:

- If you specify 'populate -lock', then by default the populate operation also uses the '-from local' option. The result is that the populate operation locks the object in the vault and keeps local modifications in your workspace. See the -from option for information.

-lock -reference (Module-based)

-lock -reference

Use the -lock option with the -reference option to populate with locked references. Locked references are useful if you intend to generate objects and want to lock them before regenerating, as opposed to editing the previous version. Upon generation of the objects, they automatically become locked copies rather than locked references. Obtaining locked references for generated objects is faster because DesignSync does not fetch the previously generated objects. If the objects exist already in the workspace, DesignSync deletes them. If the objects exist and are locally modified, the operation fails. If you intend to overwrite the modifications, use -force to create the locked references. If the default fetch state is 'reference' and you specify the -lock option without the -reference option, DesignSync leaves locked copies of the objects in your workspace; you must explicitly apply the -reference option with the -lock option if you want locked references in your workspace.

The -lock -reference combination of option is mutually exclusive with the fetch modes: -get, -share, and -mirror, and with the -recursive option.

Note: You should not use the -reference option with Cadence data collection objects. When the -reference option is used on Cadence collections, DesignSync creates a reference in the metadata for the collection object but member files are not processed and are not included in the metadata.

-lock -reference (File-based)

-lock -reference

Use the -lock option with the -reference option to populate with locked references. Locked references are useful if you intend to generate objects and want to lock them before regenerating, as opposed to editing the previous version. Upon generation of the objects, they automatically become locked copies rather than locked references. Obtaining locked references for generated objects is faster because DesignSync does not fetch the previously generated objects. If the objects exist already in the workspace, DesignSync deletes them. If the objects exist and are locally modified, the operation fails. If you intend to overwrite the modifications, use -force to create the

locked references. If the default fetch state is 'reference' and you specify the -lock option without the -reference option, DesignSync leaves locked copies of the objects in your workspace; you must explicitly apply the -reference option with the -lock option if you want locked references in your workspace.

The -lock -reference combination of option is mutually exclusive with the fetch modes: -get, -share, and -mirror.

Note: You should not use the -reference option with Cadence data collection objects. When the -reference option is used on Cadence collections, DesignSync creates a reference in the metadata for the collection object but member files are not processed and are not included in the metadata.

-log

-log <filename>	Specify the name of the populate log file. If the filename doesn't exist, DesignSync creates it. If the file does exist, DesignSync appends the new information to the end of the log file. The filename can be specified with an absolute or relative path. If you specify a path for the log file, the directory you specify must already exist and you must have write permissions to the directory in order for the log to be placed into it, DesignSync does not create the path.

-mcachemode (Module-based)

-mcachemode <mcache_mode>	Specifies how the populate command fetches the module from the module cache. Note: The module cache should always be populated at the workspace root directory level. Available modes are: o link - For each module it finds in the module cache, the populate command sets up a symbolic link from your work area to the base directory of the module in the module cache. This is the default mode on UNIX platforms. Note:

```
      - This mode is supported on UNIX platforms
        only. If you specify link mode on a
        Windows platform, the populate operation
        fails.
      -  You cannot create mcache links to
        dynamically fetched modules since there is
        no auto-refresh of mcaches.
          the populate command.

o server - Causes the populate command to
  fetch modules as physical copies from the
  server, not the module cache. (Default for
  Windows.)

The -mcachemode option overrides the default
module cache mode registry setting. If
-mcachemode is not specified, the populate
command uses the mode specified in the registry
setting. If no registry setting is specified,
the command uses link mode on Unix platforms and
server mode on Windows platforms.

Notes on mcaches:
o If you run a populate with the -norecursive
  option, the module must have been fetched into
  the mcache in -norecursive mode as well, or
  the command will not create links to or copies
  from the module cache.

o If the populate command is run using a filter,
  no mcache link to or copies are made.
  Therefore a filtered module can never be used
  in an mcache even if populate is run in a
  workspace that uses the same filter.

o The mcache administrator can fetch modules
  into a module cache to link to or copy the
  contents of the module.

o You cannot create mcache links to mcache
  directories containing members of more than
  one module version.

If a request to link to the module cache is
disallowed, DesignSync fetches the module from
the server instead.

For more information using populate with a
module cache, see 'Module Caches' in the
description section of the populate command.
```

-mcachemode (Legacy-based)

```
 -mcachemode          Specifies how the populate command fetches
```

```
<mcache_mode>        the legacy module from the module cache.

                     Note: The module cache should always be
                     populated at the workspace root directory level.

                     Available modes are:

                     o link - For each module it finds in the module
                       cache, the populate command sets up a symbolic
                       link from your work area to the base directory
                       of the module in the module cache. This is
                       the default mode on UNIX platforms.

                       Note:
                         - This mode is supported on UNIX platforms
                           only. If you specify link mode on a
                           Windows platform, the populate operation
                           fails.

                          -  You cannot create mcache links to
                             dynamically fetched modules since there is
                             no auto-refresh of mcaches.

                     o copy - For each module it finds in the module
                        cache, the populate command copies the module
                       to your work area. (Default on Windows
                        platforms)

                     o server - Causes the populate command to
                       fetch modules as physical copies from the
                       server, not the module cache.

                     The -mcachemode option overrides the default
                     module cache mode registry setting. If
                     -mcachemode is not specified, the populate
                     command uses the mode specified in the registry
                     setting. If no registry setting is specified,
                     the command uses link mode on Unix platforms and
                     copy mode on Windows platforms.

                     Notes on module mcaches:
                     o The mcache administrator can fetch legacy
                       modules into a legacy module cache to link to
                       or copy the contents of the module.

                     o Legacy modules can be fetched into either a
                       module cache or a legacy module cache by the
                       mcache administrator, but they cannot be
                       linked to or copied from.

                     If a request to link to or copy from the module
                     cache is disallowed, DesignSync fetches the
                     module from the server instead.
```

-mcachepaths (Module / Legacy-based)

-mcachepaths

Identifies one or more module caches to be searched for modules.

Path names can be absolute or relative. You can specify multiple paths in a list of space-separated path names. To specify multiple paths, surround the path list with double quotation marks ("") and separate path names with a space. For example:"/dir/cacheA /dir2/cacheB".

The path list can contain both module and legacy module mcache paths. For a module cache the path to the root directory of the module cache must be supplied.

This option overrides the default module cache paths registry setting. If -mcachepaths is not specified, the command uses the list of paths specified in the registry setting. If no registry setting is specified, the populate command fetches modules from the server.

Note:
o To specify a path that includes spaces:
  - In stcl or stclc, surround the path containing the spaces with curly braces. For example:
    "/dir1/cache {/dir2/path name}"
  - In dss or dssc, use backslashes (\) to 'escape' the spaces. For example:
    "/dir1/cache /dir2/path\ with\ spaces"
o The populate command searches the mcache in the order specified with the -mcachepaths option or in the default module cache paths registry setting if this option is absent.

-[no]merge (Module-based)

-[no]merge

Indicates whether to populate with the Latest versions from the branch specified by the persistent selector list and merge them with the current, locally modified versions. The default value is -nomerge.

If you are not doing an overlay merge (see -overlay) and the current version is not locally modified, the -merge defaults to a -get and fetches the new version without merging. By definition, a merge expects a locally modified object, so the -force option is not required.

The -merge option supports the merging work model (as opposed to the locking work model) where multiple team members can check out and edit the Latest version concurrently. The first team member to check in creates the next version. Other team members must merge the new Latest version into their local copy before they can check in their changes.

If there are no merge conflicts, the merge succeeds, leaving the merged files in your work area. If there are conflicts, DesignSync issues a warning, and you must edit the merged file to resolve the conflicts before DesignSync allows you to check in the merged version. Conflicts are shown as follows:

```
<<<<<<< local
Lines from locally modified version
=======
Lines from selected server version
>>>>>>> versionID
```

DesignSync considers the conflicts resolved when the file no longer contains any of the conflict delimiters (exactly 7 less-than, greater-than, or equal signs starting in column 1). The status of an object, as displayed by ls or from the List View in the DesignSync graphical interface, indicates if conflicts exist. The url inconflict command also determines whether a file has conflicts.

Most merges are between two versions on the same branch (the current branch and the branch specified by the persistent selector list are typically the same). However, a merge can also be performed across branches by setting the persistent selector list to a different branch. Following the merge, you are on the branch associated with the version specified by the persistent selector list (a 'merge to' operation). If you want to stay on the current branch instead, use the -overlay option. Overlay ('merge from') merges are more common when merging branches. See the -overlay option for details.

Note:

- When merging modules across branches, you should use -merge -overlay. For details about merging modules across branches, see the "Merging Across Branches section."
- The -merge option implies -get, but you can

also explicitly specify -get. For general DesignSync objects, the -merge option is mutually exclusive with all other state options (-lock, -share, -mirror, -reference, and -lock -reference).
You can use -lock with -merge for modules and their members.

o The -merge and -version options are mutually exclusive unless you specify '-version Latest'.

-merge (File-based)

-[no]merge

Indicates whether to populate with the Latest versions from the branch specified by the persistent selector list and merge them with the current, locally modified versions. The default value is -nomerge.

If you are not doing an overlay merge (see -overlay) and the current version is not locally modified, the -merge defaults to a -get and fetches the new version without merging. By definition, a merge expects a locally modified object, so the -force option is not required.

The -merge option supports the merging work model (as opposed to the locking work model) where multiple team members can check out and edit the Latest version concurrently. The first team member to check in creates the next version. Other team members must merge the new Latest version into their local copy before they can check in their changes.

If there are no merge conflicts, the merge succeeds, leaving the merged files in your work area. If there are conflicts, DesignSync issues a warning, and you must edit the merged file to resolve the conflicts before DesignSync allows you to check in the merged version. Conflicts are shown as follows:

```
<<<<<<< local
Lines from locally modified version
=======
Lines from selected server version
>>>>>>> versionID
```

DesignSync considers the conflicts resolved when the file no longer contains any of the conflict delimiters (exactly 7 less-than,

greater-than, or equal signs starting in column 1). The status of an object, as displayed by ls or from the List View in the DesignSync graphical interface, indicates if conflicts exist. The url inconflict command also determines whether a file has conflicts.

Most merges are between two versions on the same branch (the current branch and the branch specified by the persistent selector list are typically the same). However, a merge can also be performed across branches by setting the persistent selector list to a different branch. Following the merge, you are on the branch associated with the version specified by the persistent selector list (a 'merge to' operation). If you want to stay on the current branch instead, use the -overlay option. Overlay ('merge from') merges are more common when merging branches. See the -overlay option for details.

Note:
- The -merge option implies -get, but you can also explicitly specify -get. For general DesignSync objects, the -merge option is mutually exclusive with all other state options (-lock, -share, -mirror, -reference, and -lock -reference).
  You can use -lock with -merge for modules and their members.
- The -merge and -version options are mutually exclusive unless you specify '-version Latest'.

-mirror (File-based)

-mirror

Create symbolic links from the work area to objects in the mirror directory. This option requires that you have associated a mirror directory with your work area (see the 'setmirror' command).

For performance reasons, links are created only when objects do not exist in your work area. To update mirror links for existing objects, use -unifystate with the -mirror option. For example:

```
populate -recursive -full -unifystate -mirror
```

The -unifystate option does not affect locally modified objects or objects that are not part of the configuration. Use -force with

-unifystate to update the links, replacing locally modified objects and removing objects that are not part of the current configuration.

When used with the -mirror option, the -noemptydirs option does not populate directories that are empty on the mirror. Using the -force option with the -noemptydirs option removes all empty directories from the workspace. Using -force with -emptydirs for 'populate -mirror', however, populates empty directories that exist in the mirror.

The -mirror option is mutually exclusive with the other fetch modes: -lock, -get, -share, and -reference. The -mirror option is also mutually exclusive with the -keys and -from options. The -mirror option cannot take an exclude filter. If the -exclude option is specified with the -mirror fetch mode, the populate silently ignores the -exclude option.

Note:
- This option is not supported on Windows platforms.
- The -exclude option is ignored if it is included in a 'populate -mirror' operation.
- If you specify -mirror, an incremental populate does not necessarily fetch new objects checked in, nor remove links to objects deleted by team members until after the mirror is updated.
- When populating a custom generic collection from a mirror, always use 'populate -mirror' from the folder containing the collection object or from a folder above the folder containing the object.

-modulecontext (Module-based)

-modulecontext Identifies the module to be populated. Use the -modulecontext option if your workspace has overlapping modules, so that you can indicate which module to populate.

You can use the -modulecontext option when specifying a folder to populate. In this case, the populate operation filters the folder, populating only those objects that belong to the module specified with the -modulecontext option. Use -modulecontext in a recursive populate to fetch members of the specified module throughout a hierarchy.

You can also use -modulecontext option to

identify which module to fetch items from when requesting an object that is not currently in the module.

Specify an existing workspace module using the module name (for example, Chip) or a module instance name (for example, Chip%0). You also can specify -modulecontext as a server module URL (sync://server1:2647/Modules/Chip).

Notes:
o You cannot use a -modulecontext option to operate on objects from more than one module; the -modulecontext option takes only one argument, and you can use the -modulecontext option only once on a command line.

o If you have overlapping modules, you must specify -modulecontext when populating a module that contains files not present in your workspace.

-[no]new (Module-based)

-[no]new — Specifies whether to fetch module objects that are not yet in the workspace.

Apply the -new (default) to fetch all specified module objects (except those filtered out by options such as -filter and -exclude). Specify -nonew option to update only those objects already in the workspace.

Using -new is another form of filtering. It can cause the subsequent populate to be a full rather than an incremental populate.

Note: This option is supported for module objects only.

-overlay

-overlay <selectors> — Replace your local copy of the module or DesignSync non-module object with the versions specified by the selector list (typically a branch tag). The current-version status, as stored in local metadata, is unchanged. For example, if you have version 1.5 (the Latest version) of the module or DesignSync object and you overlay version 1.3, your current version is still 1.5. You could then check in this overlaid version. This operation is equivalent

to checking out version 1.3, then using 'ci -skip' to check in that version.

The behavior of the overlay operation depends on the presence of a local version and the version you want to overlay:

- If both the local version and the overlay version exist, the local version is replaced by the overlay version.
- If there is no local version but an overlay version exists, DesignSync creates a local copy of the overlay version.
- If a local version exists but there is no overlay version, the local version is unaffected by the operation.
- If the overlay version was renamed or removed, the local object is not changed, but metadata is added to it, indicating the change. This information can be viewed using the ls command with the -merged option.

Typically, you use -overlay with -merge to merge the two versions instead of overlaying one version onto another. The combination of -overlay and -merge lets you merge from one branch to another, the recommended method for merging across branches. Following the overlay merge, you are working on the same branch as before the operation.

You specify the version you want to overlay as an argument to the -overlay option. The -overlay and -version options are mutually exclusive. The -version option always updates the 'current version' information in your work area, which is not correct for an overlay operation.

- To use -overlay to specify a branch, specify both the branch and version as follows: '<branchtag>:<versiontag>', for example, 'Rel2:Latest'. You can also use the shortcut, '<branchtag>:', for example "Rel2:". If you don't explicitly specify the branch selector in this way, DesignSync does not resolve the selector as a branch selector.

When doing an overlay (with or without -merge), a number of combinations for the state of a module or DesignSync object on the two branches must be considered. For more information, see the "Merging Across Branches" section above. Hierarchical references in modules are not updated during an overlay.

Notes:

```
o The -overlay option implies -get, but
  you can also explicitly specify -get.
o The -overlay option is mutually exclusive
  with the other state options (-mirror,
  -share, -lock, -reference) and -version.
```

-path (Module-based)

```
-path <path>        Specify the name of an alternate local folder
                    to populate instead of the current folder. The
                    populate command uses the vault and persistent
                    selector list associated with the specified
                    folder.
                    Note: Using -path is equivalent to changing
                          folders, executing the populate
                          command, then changing back to the
                          original folder.

                    If you specify a folder using the -path option
                    but the folder does not exist, DesignSync
                    - verifies that a corresponding vault exists
                    - creates the folder
                    - populates the specified folder, creating
                    any interim folders necessary to replicate
                    the vault hierarchy locally.

                    If you specify the -target option but the
                    folder does not exist, DesignSync creates
                    the folder.
                    Generally, however, if the vault does not
                    exist, DesignSync does not create the
                    folder and leaves the workspace unchanged.

                    Tip: When populating a workspace with links to a
                    module cache, use -path to create the directory,
                    rather than specifying an existing directory.

                    The -path option used to be the -dir option.
                    The -dir option is still provided for backwards
                    compatibility, but is not documented
                    separately.
```

-path (Legacy-based)

```
-path <path>        Specify the name of an alternate local folder
                    to populate instead of the current folder. The
                    populate command uses the vault and persistent
                    selector list associated with the specified
                    folder.
                    Note: Using -path is equivalent to changing
                          folders, executing the populate
                          command, then changing back to the
```

```
      original folder.

If you specify a folder using the -path option
but the folder does not exist, DesignSync
- verifies that a corresponding vault exists
- creates the folder
- populates the specified folder, creating
any interim folders necessary to replicate
the vault hierarchy locally.

If you specify the -target option but the
folder does not exist, DesignSync creates
the folder.
Generally, however, if the vault does not
exist, DesignSync does not create the
folder and leaves the workspace unchanged.

Note: If the folder specified by -path does not
exist, but corresponds to a vault with
unpopulated legacy modules or DesignSync
REFERENCES, DesignSync has no way to resolve
these mappings. In this case, populate does not
create the specified folder, leaving the
workspace unchanged.

The -path option used to be the -dir option.
The -dir option is still provided for backwards
compatibility, but is not documented
separately.
```

-path (File-based)

```
-path <path>        Specify the name of an alternate local folder
                    to populate instead of the current folder. The
                    populate command uses the vault and persistent
                    selector list associated with the specified
                    folder.
                    Note: Using -path is equivalent to changing
                          folders, executing the populate
                          command, then changing back to the
                          original folder.

                    If you specify a folder using the -path option
                    but the folder does not exist, DesignSync
                    - verifies that a corresponding vault exists
                    - creates the folder
                    - populates the specified folder, creating
                    any interim folders necessary to replicate
                    the vault hierarchy locally.

                    If you specify the -target option but the
                    folder does not exist, DesignSync creates
                    the folder.
                    Generally, however, if the vault does not
```

exist, DesignSync does not create the folder and leaves the workspace unchanged.

Note: If the folder specified by -path does not exist, but corresponds to a vault with unpopulated DesignSync REFERENCES, DesignSync has no way to resolve these mappings. In this case, populate does not create the specified folder, leaving the workspace unchanged.

The -path option used to be the -dir option. The -dir option is still provided for backwards compatibility, but is not documented separately.

-[no]recursive (Module-based)

-[no]recursive — Specifies whether to perform this operation on the specified folder or module only (default), or to traverse its subfolders or submodules.

If you invoke 'populate -recursive' and specify a folder, populate operates on the folder in a folder-centric fashion, fetching the objects in the folder and its subfolders.
If the folders or subfolders contain modules or module members, populate fetches the objects, but does not follow hierarchical references (hrefs). To filter the set of objects on which to operate, use the -filter or -exclude options.

If you invoke 'populate -recursive' and specify a module, populate operates on the specified module in a module-centric fashion, fetching all of the objects in the module and following its hierarchical references (hrefs) to fetch its referenced submodules. To filter the objects on which to operate, use the -filter or -hreffilter options.

Note: Because of the way module merge handles hierarchical reference, you cannot specify -recursive when doing a cross branch merge on a module, (pop -merge -overlay).

If you invoke 'populate -recursive' on a subfolder of a module and provide a -modulecontext, populate recurses within the specified folder, fetching any object which is a member of the named module or one of its referenced submodules.
Note: For modules, you cannot use the -recursive option with the -lock option.

Note: The populate operation might skip

subfolders and individual managed objects
if their persistent selector lists differ
from the top-level folder being populated;
see the Description section for details.

If you specify -norecursive when operating on
a folder, DesignSync operates only on objects in
the specified folder. In this case, populate
does not traverse the vault folder hierarchy.
Likewise, if you specify -norecursive when
operating on a module, DesignSync operates
only on the module objects and does not follow
hrefs.

-[no]recursive (Legacy-based)

-[no]recursive

Specifies whether to perform this operation on
the specified folder only (default), or to
traverse its subfolders or hierarchical
references.

If you invoke 'populate -recursive' and specify a
folder, populate operates on the folder in a
folder-centric fashion, fetching the objects in
the folder and its subfolders. It does not follow
the hierarchical references (hrefs). To filter
the set of objects on which to operate, use the
-exclude option.

Note: The populate operation might skip
subfolders and individual managed objects
if their persistent selector lists differ
from the top-level folder being populated;
see the Description section for details.

If you specify -norecursive when operating on
a folder, DesignSync operates only on objects in
the specified folder. In this case, populate
does not traverse the vault folder hierarchy.

If you perform a -norecursive populate, then for
the subsequent populate DesignSync performs a
full populate even if the -full option is not
specified.

Notes:
- DesignSync cannot perform an incremental
  populate following a nonrecursive populate,
  because it cannot ensure that the objects in
  the work area subfolders are up-to-date.

- The -nomodulerecursive option is no longer
  required. If you apply the -nomodulerecursive
  option to legacy modules, populate recurses

within the legacy module's folders. It does not traverse REFERENCEs or hrefs of legacy modules.

-[no]recursive (File-based)

-[no]recursive

Specifies whether to perform this operation on the specified folder (default), or to traverse its subfolders.

If you invoke 'populate -recursive' and specify a folder, populate operates on the folder in a folder-centric fashion, fetching the objects in the folder and its subfolders. To filter the set of objects on which to operate, use the -exclude option.

Note: The populate operation might skip subfolders and individual managed objects if their persistent selector lists differ from the top-level folder being populated; see the Description section for details.

If you specify -norecursive when operating on a folder, DesignSync operates only on objects in the specified folder. In this case, populate does not traverse the vault folder hierarchy.

If you perform a -norecursive populate, then for the subsequent populate DesignSync performs a full populate even if the -full option is not specified.

Note: DesignSync cannot perform an incremental populate following a nonrecursive populate, because it cannot ensure that the objects in the work area subfolders are up-to-date.

-reference

-reference

Populate with DesignSync references to objects in the vault. A reference does not have a corresponding file on the file system but does have local metadata that makes the reference visible to Synchronicity programs. Populate with references when you want your work area to reflect the contents of the vault but you do not need physical copies. Use the -reference option with the -lock option to populate with locked references. Locked references are useful if you intend to generate objects and want to lock them before regenerating,

as opposed to editing the previous versions.

Note: You should not use the -reference option with Cadence data collection objects. When the -reference option is used on Cadence collections, DesignSync creates a reference in the metadata for the collection object but member files are not processed and are not included in the metadata.

-[no]replace (Module-based)

-[no]replace

This option determines how to handle locally modified objects when synchronizing your work area.

The -replace option specifies that the populate operation updates locally unmodified workspace objects. This option leaves intact all managed objects that are not members of the module (if applicable) and all unmanaged objects. If an object has been removed from the version being fetched as a result of a remove operation or retired on the server, -replace removes the member from the workspace if it has not been locally modified. (Default)

The -noreplace option specifies that the populate operation updates managed objects that have not been locally modified. The -noreplace option leaves intact all unmanaged objects. If an object has been removed from the version being fetched as a result of a remove, mvmember, rmhref or any other similar operation, -noreplace does not remove the corresponding file in the workspace.

During a recursive populate, -noreplace leaves intact managed objects belonging to a referenced submodule even when the href has been removed. If the href has been changed to reference a different submodule, -noreplace:

- Leaves intact managed objects that belong to the previous submodule but not to the new submodule
- Replaces managed members that belong to both modules with the version belonging to the new module

Notes:

- See "Forcing, Replacing, and Non-Replacing Modes" above to see how the -force option interacts with the -[no]replace option.

o If you use populate -version to populate a directory containing a module, DesignSync uses the -noreplace option unless -replace is explicitly specified.
o If you apply the -filter or -hreffilter options, populate applies the -[no]replace option on the filtered data.
o With a recursive operation, populate applies -replace and -noreplace behaviors to the top-level module and then to each referenced submodule.

-[no]replace (File-based)

-[no]replace

This option determines how to handle locally modified objects when synchronizing your work area.

The -replace option specifies that the populate operation updates locally unmodified workspace objects. This option leaves intact all managed objects and all unmanaged objects. If an object has been removed from the vault being fetched as a result of a retire, rmvault, or any other similar operation, -replace removes the file from the workspace if it has not been locally modified.

The -noreplace option specifies that the populate operation updates managed objects that have not been locally modified. The -noreplace option leaves intact all unmanaged objects. If an object has been removed from the vault being fetched as a result of a retire, rmvault, or any other similar operation, -noreplace does not remove the corresponding file in the workspace. (Default)

Notes:
o See "Forcing, Replacing, and Non-Replacing Modes" above to see how the -force option interacts with the -[no]replace option.
o If you use populate -version to populate a directory containing a module, DesignSync uses the -noreplace option unless -replace is explicitly specified.
o If you apply the -filter or -hreffilter options, populate applies the -[no]replace option on the filtered data.
o With a recursive operation, populate applies -replace and -noreplace behaviors to the top-level module and then to each referenced submodule.

-report

-report error| brief|normal| verbose	Specifies the amount and type of information displayed by the command. The information each option returns is discussed in detail in the "Understanding the Output" section above. error - lists failures, warnings, and success failure count. brief - lists failures, warnings, module create/remove messages, some informational messages, and success/failure count. normal - includes all information from brief, and lists all the updated objects, and messages about objects excluded by filters from the operation. (Default) verbose - provides full status for each object processed, even if the object is not updated by the operation.

-[no]retain

-[no]retain	Indicates whether to retain the 'last modified' timestamp of the fetched objects as recorded when each object was checked into the vault. If the workspace is set to use a mirror, or the populate is run using -share, this will also apply to the object placed in the mirror or LAN cache if the object doesn't already exist in the mirror or cache. The links in your work area to the cache or mirror have timestamps of when the links were created. If you specify the -reference option, no object is created in your work area, so there is no timestamp information at all. If an object is checked into the vault and the setting of the -retain option is the only difference between the version in the vault and your local copy, DesignSync does not include the object in populate operations. If you do not specify '-retain' or -noretain', the populate command follows the DesignSync registry setting for Retain last-modification timestamps. By default, this setting is not enabled; therefore, the timestamp of the local object is the time of the populate operation. To change the default setting, your

Synchronicity administrator can use the SyncAdmin tool. For information, see SyncAdmin Help.

The mirror system, by default,fetches objects into the mirror with the -retain option. The mirror administrator, however, can define mirrors to use the -noretain option. The default setting should agree with the Retain last-modification timestamp registry setting to maintain consistency. See the "Mirror Administration Server Registry Settings" topic for setting of the co or populate options for mirrors.

Note: When fetching from the cache or mirror (by specifying the '-from local' option), the last modified timestamp comes from the file in the cache or mirror, not from the version that was checked into the vault. If the file was fetched into the cache or mirror with the -retain option, these two timestamps are the same. But if the file was fetched into the cache or mirror with the -noretain option and then fetched into the workspace with both the '-from local' and '-retain' options, the 'last modified' timestamp used is the time the object was fetched into the cache or mirror.

-savelocal

-savelocal <value> This option affects collections that have local versions.

When it fetches an object, the populate operation first removes from your workspace any local version that is unmodified. (To remove a local version containing modified data, specify 'pop -force'.) Then the populate operation fetches the object you are checking out (with the local version number it had at the time of checkin).

The -savelocal option specifies the action that the populate operation takes with modified local versions in your workspace (other than the current, or highest numbered, local version). (DesignSync considers a local version to be modified if it contains modified members or if it is not the local version originally fetched from the vault when the collection object was checked out or populated to your workspace.)

Specify the -savelocal option with one of the following values:

save - If your workspace contains a local version other than the local version being fetched, the populate operation saves the local version for later retrieval. See the 'localversion restore' command for information on retrieving local versions that were saved.

fail - If your workspace contains an object with a local version number equal to or higher than the local version being fetched, the populate operation fails. This is the default action.

Note: If your workspace contains an object with local version numbers lower than the local version being fetched and if these local versions are not in the DesignSync vault, the populate operation saves them. This behavior occurs even when you specify '-savelocal fail'

delete - If your workspace contains a local version other than the local version being fetched, the populate operation deletes the local version from your workspace.

If you do not specify the -savelocal option, the populate operation follows the DesignSync registry setting for SaveLocal. By default, this setting is "Fail if local versions exist" ('-savelocal fail'). To change the default setting, a Synchronicity administrator can use the Command Defaults options pane of the SyncAdmin tool. For information, see SyncAdmin Help.

Note:
- You may need to use the -force option with the -savelocal option to allow the object being fetched to overwrite a locally modified copy of the object. For an example scenario, see EXAMPLES.
- The -savelocal option affects only objects of a collection defined by the Custom Type Package (CTP). This option does not affect objects that are not part of a collection or collections that do not have local versions.

-share

-share

Fetch shared copies. Shared objects are stored in the file cache directory and links to the

```
                        cached objects are created in the work area.

                        Notes:
                        This option is not supported on Windows
                        platforms.

                        The -share option is mutually exclusive with the
                        other fetch modes: -lock, -get, -mirror, and
                        -reference.  The -share option is also mutually
                        exclusive with the -keys and -from options.
```

-target (Legacy-based)

```
 -target                Specifies a legacy module configuration to fetch
 <server_module_url>    to your work area. Note: This option applies
                        only to legacy modules. Also, this option is
                        no longer required and will be removed in a
                        future release; you can specify the module
                        as a command argument. See ARGUMENTS above
                        to specify the module as an argument.

                        To specify a module using the -target option,
                        use the syntax:
                           sync[s]://<host>[:<port>]/<vaultPath>
                        where <host> is the SyncServer on which the
                        module resides, <port> is the SyncServer port
                        number, and <vaultPath> is the vault folder in
                        which the module's data resides.

                        To specify a module configuration other than
                        the default configuration, use the syntax:
                        sync[s]://<host>[:<port>]/<vaultPath>@<config>
                        where <host> is the SyncServer on which the
                        module resides, <port> is the SyncServer port
                        number, <vaultPath> is the vault folder in
                        which the module's data resides, and <config>
                        is the specific configuration of the module.

                        If you specify this option, the populate
                        command sets the vault and selector.

                        If you specify the 'populate -target' with the
                        -path option and the specified directory does
                        not exist, the populate command creates the
                        directory in your work area and sets the
                        selector for fetching the configuration
                        specified with '-target'.

                        Note: To fetch an entire legacy module
                        hierarchy, use the -recursive option with
                        'populate -target'.

                        The 'populate -target' command checks whether
                        the target is an ordinary DesignSync vault or a
```

module with no hrefs. In the cases where it is either a DesignSync Vault or a module with no hrefs and the registry setting indicates that the module with no hrefs should be treated like a DesignSync vault, it performs a setvault operation with the value specified to target and then performs an ordinary populate on the directory. Effectively, this is equivalent to performing a 'setvault' and populate (without -target). The setvault is done recursively if the -recursive option was specified with populate.

-trigarg

-trigarg <arg>

Specifies an argument to be passed from the command line to the triggers set on the populate operation. If the argument contains whitespace, enclose the argument within double quotation marks ("") if using the dss command shell or braces ({}) if using the stcl command shell.

-[no]unifystate

-[no]unifystate

Indicates whether to set the state of all objects processed, even up-to-date objects, to the specified state (-get, -lock, -share, -mirror, or -reference) or to the default fetch state if no state option is specified. See the "fetch preference" help topic for more information.

By default, populate changes the state of only those objects that are not up-to-date (-nounifystate). If the -unifystate option is specified, DesignSync changes the state of the up-to-date objects, as well, and thus performs a full populate.

The -unifystate option does not change the state of locally modified objects; use -force with -unifystate to force a state change, thus overwriting local modifications. The -unifystate option does not change the state of objects not in the configuration; use -force with -unifystate to remove objects not in the configuration.
The -unifystate option does not cancel locks; you can check in the locked files, use the 'cancel' command to cancel locks you have acquired, or use the 'unlock' command to cancel

team members' locks.

Note: The -unifystate option is ignored when you lock design objects. If you populate with locked copies or locked references, DesignSync leaves all processed objects in the requested state.

-version (Module-based)

-version <selector> Specifies the versions of the objects to populate. The selector list you specify (typically a version or branch tag) overrides the persistent selector lists of the objects you are populating. If you populate the top-level module in a hierarchy with the -version tag, you replace the persistent selector of the workspace with the version specified by this option. If you specify the -recursive option, the specified selector list is used to populate all subfolders during populates.

If you specify a date selector (Latest or Date(<date>)), DesignSync augments the selector with the persistent selector list to determine the versions to populate. For example, if the persistent selector list is 'Gold:,Trunk', and you specify 'populate -version Latest', then the selector list used for the populate operation is 'Gold:Latest,Trunk:Latest'.

For details on selectors and selector lists see the topic describing selectors.

Note:
- Using the -version option with the populate command changes the workspace selector if the populate was performed on a top-level module instance. If you are working in a module hierachy, you should use the swap replace command to change the sub-module version populated. If you populate individual module members or folders, the persistent selector is not updated.
- If you use -version to populate a module member, populate fetches the version that is appropriate to the module version as identified by the version value.
- If you use the -version option with the -incremental option, and the selector you specify does not exactly match the workspace selector, the incremental populate does not occur. DesignSync performs a full populate

instead. See "Incremental Versus Full Populate" in the description section for more information.
- When using -version to specify a branch, specify both the branch and version as follows: <branchtag>:<versiontag>, for example, Rel2:Latest. You can also use the shortcut, <branchtag>:, for example Rel2:. If you do not explicitly specify the branch selector in this way, DesignSync does not resolve the selector as a branch selector.
- When you specify a version-extended name that reflects the object's version, for example, "file.txt;1.3", populate ignores the -version option.
- Specify '-version <branchtag>:Latest' only if necessary. In some cases, DesignSync augments the selector to be <branchtag>:Latest. When you append ':Latest', it may not match the work area selector. This mismatch invalidates your next incremental populate resulting in a slower, full populate.
- The -version option is mutually exclusive with -merge unless you specify '-version Latest', the default.
- The -version and -overlay options are mutually exclusive.

-version (File / Legacy-based)

`-version <selector>` Specifies the versions of the objects to populate. The selector list you specify (typically a version or branch tag) overrides the persistent selector lists of the objects you are populating. If you specify the -recursive option, the specified selector list is used to populate all subfolders during populate.
You can also specify a ProjectSync configuration; see "Interaction with Legacy Modules" in the Description section.

If you specify a date selector (Latest or Date(<date>)), DesignSync augments the selector with the persistent selector list to determine the versions to populate. For example, if the persistent selector list is 'Gold:,Trunk', and you specify 'populate -version Latest', then the selector list used for the populate operation is 'Gold:Latest,Trunk:Latest'.

For details on selectors and selector lists see the topic describing selectors.

Note:
o Using the -version option with the populate command does not change the workspace selector, even during the initial populate of an object. To set the workspace selector as part of the populate command, specify the selector explicitly, using the <object>;<selector> syntax.
o If you use the -version option with the -incremental option, and the selector you specify does not exactly match the workspace selector, the incremental populate does not occur. DesignSync performs a full populate instead. See "Incremental Versus Full Populate" in the description section for more information.
o When using -version to specify a branch, specify both the branch and version as follows: <branchtag>:<versiontag>, for example, Rel2:Latest. You can also use the shortcut, <branchtag>:, for example Rel2:. If you do not explicitly specify the branch selector in this way, DesignSync does not resolve the selector as a branch selector.
o When you specify a version-extended name that reflects the object's version, for example, "file.txt;1.3", populate ignores the -version option.
o Specify '-version <branchtag>:Latest' only if necessary. In some cases, DesignSync augments the selector to be <branchtag>:Latest. When you append ':Latest', it may not match the work area selector. This mismatch invalidates your next incremental populate resulting in a slower, full populate.
o The -version option is mutually exclusive with -merge unless you specify '-version Latest', the default.
o The -version and -overlay options are mutually exclusive.
o When you use populate with the -version option to fetch a directory containing legacy modules, by default DesignSync uses the -noreplace

-view (Module-based)

-view view1 [,view2[,view...]]	Module view name or comma-delimited list of module view names, applied to a module or module hierarchy when it is fetched. Note: This option is only valid for server module objects. If it is used with an argument

type other than a server module url, the option is silently ignored.

There is no default value for this option. You cannot set a default value in the command defaults system.

On an initial populate, the module view name or names list provided is propagated through the hierarchy and applied to all fetched modules. The module view name or names list is also saved, or persisted in the workspace metadata for each module so that all subsequent populates use the same view. The documentation refers to a view saved in the metadata as a "persistent module view" because it persists through subsequent populates rather than needed to be specified with each command.

If a persistent module view has been set on a workspace module, any sub-modules subsequently populated use the persistent module view already defined for parent module.

Tip: Since populate calls the Checkout Access Control, you can write an Access Control filter to cause populate to fail if no module view is specified or tie users to specific module views.

Notes:

- If none of the specified module views exist on the server, DesignSync issues a warning and the populate command runs as if no view were specified. If, in a list of module views, one or more views exists, and one or more views does not exist, the populate command silently ignores the non-existent view(s).
- When the persistent module view set on the workspace is changed, the subsequent populate is a full populate. For more information on changing or clearing the persistent view, see the setview command.

-xtras (Module-based)

-xtras <list>

List of command line options to pass to the external module change management system. Any options specified with the -xtras option are sent verbatim, with no processing by the populate command, to the Tcl script that defines the external module change management system.

RETURN VALUE

```
In dss/dssc mode, you cannot operate on return values, so the return
value is irrelevant.

In stcl/stclc mode, two lists are returned, where the first list
is non-empty if at least one object was successfully processed,
and the second list is non-empty if at least one object failed.

Notes:
 - "successfully processed" may not mean "successfully populated".
   For example, a populate of an object that you already have in
    your work area is considered a success even though no checkout
   occurs.
 - Scripts should only test for non-empty lists to determine
   success or failure. The actual content of the non-empty
   lists might change in subsequent releases.
 - If all objects fail, an exception occurs (the return value is
   thrown, not returned).
 - Objects reported as "excluded by filter," may have been excluded
   either with the -filter option (for modules) or with the -exclude
   option (for any DesignSync objects.)
```

SEE ALSO

```
caching, ci, co, command defaults, localversion, remove, retire,
selectors, setselector, setvault, setview, swap, url contents
```

EXAMPLES

- Example of Populating a Module (Module-based)
- Example of Populating a Specific Module Member (Module-based)
- Example of Populating a Module with a Static Selector (Module-based)
- Example of Populating a Module Using Version-Extended Naming (Module-based)
- Example of Creating a Module Cache (Module-based)
- Example of Populating an Mcache Link (Module-based)
- Example of Populating a Module View (Module-based)
- Example of Specifying a Hierarchical Hreffilter (Module-based)
- Example of Merge Across Branches (Module-based)
- Example of Creating a new work area from a DesignSync vault (File-based)
- Example of Creating a New Work Area from a DesignSync Vault Branch (File-based)
- Example of Updating an Existing Workspace with a Full Populate (File-based)
- Example of Updating the State of Objects in the Workspace (File-based)
- Example of Performing a Merge into a Workspace (File-based)
- Example of Replacing Modified Files with the Server Versions (File-based)

Example of Populating a Module (Module-based)

```
The following example shows how to populate module Chip in the
workspace directory ~/chip.
For an initial populate, provide the server URL of the module:

stcl> pop sync://guaraldi:30077/Modules/Chip

This creates the Chip module with the current directory
as the base directory:

Beginning populate operation...

Making Module with
  Base Dir = /home/karen/chip
  Name = Chip
  URL = sync://guaraldi:30077/Modules/Chip
  Selector = Trunk:Latest

Created Module with instname Chip%1

Populating objects in Module Chip%1 with Base Dir /home/karen/chip...

/chip/makefile:  Success - Checked Out  version: 1.1
/DOC/Chip.doc:  Success - Checked Out  version: 1.1
/chip/verilog/chip.v:  Success - Checked Out  version: 1.1

Chip%1: Version of module in workspace updated to 1.2

Finished populate of Module Chip%1 with Base Dir  /home/karen/chip

Finished populate operation...

{Objects succeeded (3)} {}

When you next update your work area using the populate command,
you can supply the workspace module name or the workspace folder
name.  In the following example the workspace folder name is
supplied, and there have been no changes since the last populate:

 stcl> pop -recursive ~/chip
  Beginning populate operation at Thu Apr 19 02:16:31 PM EDT 2007...



  Populating objects in Module          Chip%1
                       Base Directory  /home/karen/chip
                       Without href recursion



  Chip%1 : Version of module in workspace retained as 1.2

  Finished populate of Module Chip%1 with base directory
  /home/karen/chip

  Finished populate operation.

   {} {}
```

Example of Populating a Specific Module Member (Module-based)

```
 The following is an example of fetching a specific version of a
 module member:

  stcl> pop -version 1.4 File1.txt

  Populating objects in Module          JitaMod1%0
                      Base Directory  /home/tachatterjee/JitaMOD
                      Without href recursion


  Fetching contents from selector '1.4', module version '1.4'

  Total data to transfer: 0 Kbytes, 1 files, 0 collections
  Progress: 0 Kbytes, 1 files, 0 collections, 100.0% complete
  /File1.txt: Success - Checked Out  version: 1.3

  Finished populate operation...

 This fetches the version of the file File1.txt contained in
 version 1.4 of the module.
```

Example of Populating a Module with a Static Selector (Module-based)

```
 The following example shows the messages you receive when you
 populate a static selector into a workspace.

 dss>  populate -recursive -version Gold Chip-R419%0
  Beginning populate operation at Fri Oct 28 12:41:08 Eastern Daylight
  Time 2016...

  Setting Selector [Gold] on workspace module
  c:\workspaces\ChipDev419\chip\Chip-R419%0
  WARNING: Chip-R419%0: Changing the selector to a static value (Gold).
  You will not be able to check in module or member modifications.

  Selector on module c:\workspaces\ChipDev419\chip\Chip-R419%0 was
  modified.

  Populating objects in Module          Chip-R419%0
                       Base Directory  c:\workspaces\ChipDev419\chip
                       With href recursion


  Fetching contents from selector 'Gold', module version '1.5.1.1'
...
  Finished populate operation.

  ##### WARNINGS and FAILURES LISTING #####
  #
  # WARNING: Chip-R419%0: Changing the selector to a static value
```

```
#(Gold).
# You will not be able to check in module or member modifications.
#
########################################

 {Objects succeeded (6)} {Objects failed (0)}
```

Example of Populating a Module Using Version-Extended Naming (Module-based)

```
The following example shows how to fetch a specific version of a
module using a version-extended name.

In this example, the latest version of the file is 1.5. You can
do a vhistory to determine which version of the file you want to
fetch.

To fetch version 1.2 of the file:

 stcl> pop "File1.txt;1.2"

 Beginning Check out operation...

 Checking out: File1.txt          : Success - Fetched  version: 1.2

 Checkout operation finished.


 Finished populate operation...
```

Example of Creating a Module Cache (Module-based)

```
The following example shows how to populate a module cache using the
-share option to create a copy of the module in a centralized
location.

Note:  The module cache directory must be writable by the
creator/owner of the module cache, but not by the users of the module
cache.

stcl> populate -share -
```

Example of Populating an Mcache Link (Module-based)

```
The following example shows how to populate module Chip
using the -mcachepaths option to fetch contents from the module
cache named 'designs' located in the mcacheDir directory.

  stcl> populate -get -recursive -hrefmode static
   -path /home/rsmith/MyModules/designs -mcachemode link -mcachepaths
  /home/mcacheDir/  sync://srv2.ABCo.com:2647/Modules/Chip/
```

```
  Beginning populate operation at Mon Jun 23 10:36:43 AM EDT 2008...

  sync://srv2.ABCo.com:2647/Modules/Chip/: : Created mcache
  symlink /home/rsmith/MyModules/designs.

  Creating Module Instance 'Chip%1' with base directory
  '/home/rsmith/MyModules/designs'

  Finished populate operation.

   {Objects succeeded (1)} {}

 Note: Any existing workspace content will not be replaced with
 module cache links.  To replace workspace content you must first
 remove from the workspace those configurations to be replaced.  Use
 the 'rmfolder -recursive' command on the configuration base
 directory, or specify a non-existent directory for the -path option
 to create a new directory for the module cache links.
```

Example of Populating a Module View (Module-based)

```
 This example shows populating a workspace with a module view list;
 specifically the the RTL and DOC Module Views.

 stcl> populate -get -view RTL,DOC -path ./Chip sync://
 srv2.ABCo.com:2647/Modules/Chip

 Beginning populate operation at Fri May 06 02:04:38 PM EDT 2011...

 Populating module instance with

 Base Directory = /users/larry/MyModules/Chip
 Name = Chip
 URL = sync:// srv2.ABCo.com:2647/Modules/Chip
 Selector = Trunk:
 Instance Name = Chip%2
 Metadata Root = / users/larry/MyModules
 View(s) = RTL,DOC

 Recursive Mode = Without href recursion

 Fetching contents from selector 'Trunk:', module version '1.9'
 Total data to transfer: 1 Kbytes (estimate), 5 file(s), 0 collection(s)

 Progress - from local cache: 0 Kbytes, 0 file(s), 0 collection(s)
 Progress - from server: 1 Kbytes, 5 file(s), 0 collection(s), 100.0%
complete

 Chip%2/makefile : Success - Checked out version: 1.2
 Chip%2/README : Success - Checked out version: 1.3
 Chip%2/doc/chip.html : Success - Checked out version: 1.2
 Chip%2/doc/chip.doc : Success - Checked out version: 1.2
 Chip%2/verilog/chip.v : Success - Checked out version: 1.5
```

```
 Chip%2/verilog/chip_inc.v : Success - Checked out version: 1.3

 Chip%2 : Version of module in workspace updated to 1.9

 Finished populate of Module Chip%2 with base directory
 /users/larry/MyModules/Chip

 Time spent: 0.2 seconds, transferred 1 Kbytes, copied from local
 cache 0 Kbytes, average data rate 4.9 Kb/sec

 Finished populate operation.

 {Objects succeeded (5)} {}
```

Example of Specifying a Hierarchical Hreffilter (Module-based)

```
 This example shows an initial populate using a hierarchical href
 filter to exclude the /BIN module from the workspace when it appears
 beneath the /JRE module. In this example, the module hierarchy is set
 up like this:
 NZ214 <- ROM <- JRE <- BIN
 With NZ214 being the top-level Chip design module.

 Note: Whenever you use the -hreffilter option, you must populate
 recursively.

 dss> populate -recursive -retain -full -hreffilter JRE/BIN
 sync://serv1.ABCo.com:2647/Modules/Chip/NZ214

  Beginning populate operation at Wed Dec 11 13:24:31 Eastern Standard
 Time 2013...

  Populating module instance with
     Base Directory  = c:\workspaces\V6R2014x\chipDesign
     Name            = NZ214
     URL             = sync://serv1.ABCo.com:2647/Modules/Chip/NZ214
     Selector        = Trunk:
     Instance Name   = NZ214%1
     Metadata Root   = c:\workspaces\V6R2014x
     Recursive Mode  = With href recursion


  Fetching contents from selector 'Trunk:', module version '1.3'

  Total data to transfer: 0 Kbytes (estimate), 6 file(s), 0 collection(s)
  Progress - from local cache: 0 Kbytes, 0 file(s), 0 collection(s)
  Progress - from local cache: 0 Kbytes, 0 file(s), 0 collection(s)

  Progress - from server: 0 Kbytes, 0 file(s), 0 collection(s), 0.0% complete
  Progress - from server: 1 Kbytes, 6 file(s), 0 collection(s), 100.0%
complete

  NZ214%1\chip.ver : Success - Checked out version: 1.1
 ...
```

```
  Creating sub module instance 'ROM%1' with base directory
 'c:\workspaces\V6R2014x\chipDesign\ROM'

 Finished populate of Module NZ214%1 with base directory
 c:\workspaces\V6R2014x\chipDesign

 Time spent: 0.3 seconds, transferred 1 Kbytes, copied from local cache 0
Kbytes, average data rate 3.4 Kb/sec

  ======================================================

  Populating sub module instance with
     Base Directory  = c:\workspaces\V6R2014x\chipDesign\ROM
     Name            = ROM
 ...
  Creating sub module instance 'JRE%0' with base directory
     'c:\workspaces\V6R2014x\chipDesign\ROM\JRE'

 Finished populate of Module ROM%1 with base directory
 c:\workspaces\V6R2014x\chipDesign\ROM

 Time spent: 0.0 seconds, transferred 0 Kbytes, copied from local
 cache 0 Kbytes, average data rate 0.0 Kb/sec

  ======================================================

  Populating sub module instance with
     Base Directory  = c:\workspaces\V6R2014x\chipDesign\ROM\JRE
 ...
  JRE%0 : Version of module in workspace updated to 1.2

  BIN :   Sub Module Excluded by Hierarchical Filter
  Finished populate of Module JRE%0 with base directory
  c:\workspaces\V6R2014x\chipDesign\ROM\JRE

  Time spent: 0.0 seconds, transferred 0 Kbytes, copied from local
  cache 0 Kbytes, average data rate 0.0 Kb/sec


  Finished populate operation.

  {Objects succeeded (8)} {}
```

Example of Merge Across Branches (Module-based)

```
 This example shows a simple module merge across branches.  After you
 perform the merge, you must check in your changes to apply the merge
 changes to the modules.

 dss> pop -merge -overlay Branch: ROM%1
  Beginning populate operation at Tue Apr 10 01:55:24 PM EDT 2007...
```

```
Populating objects in Module          ROM%1
                      Base Directory  /home/rsmith/MyModules/rom
                      Without href recursion


Fetching contents from selector 'Branch:', module version '1.3.1.3'

Merging with Version: 1.3.1.3
Common Ancestor is Version: 1.3

=========================================================================
Step 1: Identifying items to be merged and conflict situations
=========================================================================

/romMain.c : member will be fetched from merged version and
      added to workspace version on checkin.
      Use 'ls -merged added' to identify members added by merge.
/rom.v : conflict - different member in merge version found at same natural
       path in workspace version. Cannot fetch member or merge contents
       with member from merge version; it will be skipped. If member from
       merge version is desired, remove or move member on workspace
       branch and then re-populate with overlay from merge version.
/rom.v : Natural path different on merge version and workspace version.
      Contents will be merged, if required.
/rom.doc : No merge required.
/doc/rom.doc : No merge required.

=========================================================================
Step 2: Transferring data for any items to be fetched into the
workspace
=========================================================================

Total data to transfer: 0 Kbytes (estimate), 1 file(s), 0 collection(s)
Progress: 0 Kbytes, 0 file(s), 0 collection(s), 0.0% complete
Progress: 1 Kbytes, 1 file(s), 0 collection(s), 100.0% complete


=============================================================================
Step 3: Merging file contents as required into the workspace
=============================================================================


Beginning Check out operation...

Checking out: rom.v                          : Success - Version
1.1.1.1 has replaced version 1.1.
Checking out: rom.c                          : Success - Version
1.1.1.1 has replaced version 1.1.

Checkout operation finished.


=========================================================================
Step 4: Updating files fetched into the workspace
=========================================================================

/romMain.c : Success - Version 1.1 fetched
```

```
 ROM%1 : Version of module in workspace not updated (Due to overlay
 operation).


 ========================================================================
 Step 5: Comparing hrefs for the workspace version and merge version:
 ========================================================================
     No hrefs present in workspace version
     No hrefs present in merge version


 Finished populate of Module ROM%1 with base directory
 /home/rsmith/MyModules/rom

 Time spent: 0.2 seconds, transferred 1 Kbytes, average data rate 4.3 Kb/sec

 Finished populate operation.

  {Objects succeeded (3)} {}
```

After the populate has completed, run ci to create the new module version with the merge changes.

```
dss> ci -comment "Incorporating changes on Branch:" ROM%1
   Beginning Check in operation...


 Checking in objects in module ROM%1

 Total data to transfer: 1 Kbytes (estimate), 3 file(s), 0 collection(s)
 Progress: 0 Kbytes, 0 file(s), 0 collection(s), 0.0% complete
 Progress: 2 Kbytes, 3 file(s), 0 collection(s), 100.0% complete

 Checking in: /rom.c                          Success - New version: 1.2
 Checking in: /rom.v                          Success - New version: 1.2
 Checking in: /romMain.c                      Success - New version: 1.1.1.1

 ROM%1: Version of module in workspace updated to 1.5

 Finished checkin of Module ROM%1, Created Version 1.5

 Time spent: 0.7 seconds, transferred 2 Kbytes, average data rate 2.8 Kb/sec
 Checking in: /doc/rom.doc                    : Success - No new
 version created. Lock Removed.

 Checkin operation finished.

  {Objects succeeded (4)} {}
```

After the checkin has created the new module version, you can create a merge edge to store a record of the changes.

```
 dss> mkedge ROM%1
 Edge from 1.3.1.3 to 1.5 for module
 sync://srv2.ABCo.com:2647/Modules/ROM created successfully.
```

Example of Creating a new work area from a DesignSync vault (File-based)

```
The following example creates a new work area containing unlocked
copies of every object in the vault hierarchy:

  dss> scd /home/tgoss/Projects/Asic
  dss> setvault sync://myhost.myco.com:2647/Projects/Asic .
  dss> populate -recursive -get

Because -version is not specified, the persistent selector list of
the current folder determines what versions to populate. The local
Asic folder has not had a 'setselector' command applied to it or
any parent folder, so the default persistent selector list is
'Trunk'. By default, DesignSync performs an incremental populate
of the Latest versions on the specified branch (Trunk). Note that
this operation does not fetch objects whose 'Trunk' branch is
retired.
```

Example of Creating a New Work Area from a DesignSync Vault Branch (File-based)

```
The following example differs from the previous example in that the
work area is for the Rel2.1 branch, not Trunk, and the work area
contains links to a cache directory instead of local copies:

  dss> scd /home/tgoss/Projects/Asic
  dss> setvault sync://myhost.myco.com:2647/Projects/Asic@Rel2.1:Latest .
  dss> populate -recursive -share
```

Example of Updating an Existing Workspace with a Full Populate (File-based)

```
The following example performs a full (nonincremental) recursive
populate on the current folder, fetching unlocked copies of files
for updated objects. Note that the states of objects that are not
updated DO NOT change.

  dss> populate -recursive -full -get
```

Example of Updating the State of Objects in the Workspace (File-based)

```
By default, the states of up-to-date objects do not change during
a populate operation.  The following example updates the states of
the objects that are up-to-date, allowing you to unify the states
of all objects in your work area.  The -unifystate option causes
DesignSync to perform a full populate rather than an incremental
populate.

  dss> populate -recursive -unifystate -get
```

Example of Performing a Merge into a Workspace (File-based)

The following example merges Latest versions from the current branch into the local versions. You perform this operation when your team uses the merging (nonlocking) work model and you and other team members have been modifying the same objects. It is more common to use the 'co -merge' command to operate on just those objects you want to check in.

```
dss> populate -merge
```

Note that the merge operation fetches from the branch specified by the folder's persistent selector list, not from the current branch. However, these two branches are typically the same unless you have changed the persistent selector list with the setselector command. In this case, you would be merging across branches instead of from the same branch. This method for merging between two branches is not recommended; use the -overlay option.

The following example merges one branch (Dev) into another (Main). This operation is typically performed by a release engineer who manages the project vault. The work area is first populated with the Latest versions from 'Main'. Then the Latest versions from Dev are merged into the local versions. The -overlay option indicates that after the operation, the current branch and version information (as stored in local metadata) should be unchanged. Following the merge and after any merge conflicts are resolved, a check-in operation checks the merged version into 'Main'.

```
dss> url selector .
Main:Latest
dss> populate -recursive
dss> populate -recursive -merge -overlay Dev:Latest
  [Resolve any merge conflicts]
dss> ci -recursive -keep .
```

Example of Replacing Modified Files with the Server Versions (File-based)

This example shows use of the populate operation that deletes local versions.

Note: The DesignSync Milkway integration has been deprecated. This example is meant to be used only as a reference.

Mike checks out the Milkyway collection object top_design.sync.mw, which fetches local version 4 of that object to his workspace. He modifies the object and creates local version 5. Then he checks in top_design.sync.mw. The check-in operation does not remove local versions, so Mike now has local version 5 (unmodified) and local version 4 in his workspace. (Note: Because the checkin removes local version 4's link to with the original check-out operation of top_design, DesignSync now considers local version 4 to be modified.)

Ben checks out top_design.sync.mw (local version 5). He creates local version 6 and checks the object in.

Mike does some work on top_design, which creates local versions 6, 7, and 8 in his workspace. Then he decides to use Ben's version of the top_design object instead.

Mike uses populate to fetch the latest versions of Milkyway collection objects to his workspace. He doesn't want to save his local versions of the object, so he uses the '-savelocal delete' option to delete local versions other than the local version being fetched. In addition, he uses the -force option. (Because he created local versions 6, 7, and 8 of top_design in his workspace, DesignSync considers the top_design object to be locally modified and by default the populate operation does not overwrite locally modified objects. To successfully check out top_design, Mike must use '-force'.)

```
stcl> cd /home/tjones/top_design_library
stcl> populate -savelocal delete -force
```

Before fetching top_design.sync.mw from the vault, the populate operation first deletes all local versions that are unmodified. So the populate operation deletes Mike's local version 6 because that was the version originally fetched and its files are unmodified.

Because Mike specified the -force option, the populate also deletes Mike's local version 8 (the current local version containing modified data for the object).

Because Mike specified '-savelocal delete', the populate operation deletes local version 7, which is not in the vault and is not the modified data Mike agreed to delete when he specified '-force'. If Mike specified '-savelocal save', DesignSync would save local version 7. Local version 4 is also deleted.

Finally, Mike's populate operation fetches the top_design object (Ben's local version 6) from the vault.

Mike continues to modify the top_design object, creating local version 7, which he checks in.

Ben has local versions 5 and 6 in his workspace. He populates his workspace containing the top_design collection object (local version 7), specifying '-savelocal fail'. The populate operation removes local version 6 from his workspace because it is unmodified. The operation saves local version 5 even though it is modified. (Ben's checkin of local version 6 removed local version 5's link to with the original checkout of top_design, so DesignSync now considers local version 5 to be modified.) The populate also takes place despite the fact that Ben specified '-savelocal fail'. The populate operation takes this action because local version 5 has a number lower than the local version being fetched. If Ben had instead specified '-savelocal delete', the populate operation would delete local version 5.

setmirror

setmirror Command

NAME

setmirror - Maps a mirror directory to a working directory

DESCRIPTION

This command associates a local working directory with a mirror directory. The mirror directory, as conveyed to you by your project leader, will contain a set of data for your project that is automatically updated by a Mirror Administration Server (MAS) at your site to exactly mimic the data set defined for your project vault. For example, your team may always want the Latest version of files on the main Trunk branch. Another team may want a mirror for a development branch, that always contains the file versions on that branch with a specific tag.

Each site will have a MAS for its LAN, which automatically maintains the site's mirrors, based on the definition of each mirror. Once your workspace is associated with a mirror, you can use the '-mirror' option to DesignSync revision control commands. The '-mirror' option results in symbolic links in your workspace, leading to files in the mirror.

All subdirectories of the local working directory inherit the mirror location you specify with setmirror. You cannot use setmirror on a subdirectory to specify a different mirror directory than is set for the parent directory.

Note:
To resolve the mirror location, DesignSync does not search above the root of a workspace where a setvault has been applied. Thus, if a setvault has been applied to a folder (/Projects/ASIC/alu) and you apply the 'setmirror' command at a higher-level folder (for example, /Projects/ASIC), the 'setmirror' command is ignored at and below the folder where the setvault occurred (/Projects/ASIC/alu).

By default, 'setmirror' stores the path to a mirror exactly as it was specified to the 'setmirror' command. To instead have 'setmirror' resolve the path, see the "DesignSync Client Commands Registry Settings" topic in the DesignSync Data Manager User's Guide.

To remove the association between a working directory and a mirror directory (an 'unsetmirror' operation), specify an empty string ("") as the mirror directory argument. When specifying an empty string as the mirror directory argument, the 'setmirror' command must be run from within a DesignSync shell. An Operating System shell cannot pass an empty string value to a DesignSync shell.

Executing setmirror without any arguments displays the mirror

```
for the current directory, or an empty string if the current
directory has no mirror set.

Note:
Mirrors can be treated in the same way as your DesignSync
work areas. For example, you can use commands such as the url
commands or ls on mirror directories.
```

SYNOPSIS

```
setmirror [--] [<mirrorDirectory> <localWorkingDirectory>]
```

OPTIONS

- --

--

```
--          Indicates that the command should stop looking for
            command options. Use this option when arguments
            to the command begin with a hyphen (-).
```

RETURN VALUE

```
none
```

SEE ALSO

```
url mirror, populate, ci, co, cancel
```

EXAMPLES

```
This example creates an association between a local
working directory /home/goss/Projects/ASIC and the mirror
directory /users/admin/Projects/mirror/ASIC.
 dss> setmirror /users/admin/Projects/mirror/ASIC /home/goss/Projects/ASIC
 ASIC:  Success Set Mirror
 dss> scd  /home/goss/Projects/ASIC
 dss> setmirror
 file:///users/admin/Projects/mirror/ASIC
```

setroot

setroot Command

NAME

```
setroot             - Sets the root workspace location
```

DESCRIPTION

- Notes for Modules Root

```
This command designates the workspace directory used as a storage
area for a set of local metadata information for a collection of
data (module or files-based data). The metadata includes information
about the DesigSync objects,

When a DesignSync object is populated into a workspace that has not
had a root folder set for it, then the parent folder of the base
directory being populated is automatically set as the root folder.

Note: You cannot define a root folder underneath (or within) an existing
root directory.
```

Notes for Modules Root

```
After the root folder is defined and the metadata is created, you can
refer to a module by the module instance name, rather than specifying
the full module path name.
```

SYNOPSIS

```
setroot -[[un]set] [--] <workspace folder>
```

ARGUMENTS

- Workspace Folder

Workspace Folder

```
<workspace      The name of the workspace folder to designate as the
  folder>       root folder. The folder must already exist to be
                designated as the root folder.
```

OPTIONS

-[un]set (Module-based)

```
-[un]set      Indicates whether to set the workspace root or remove
              the workspace root setting from a workspace.

              -unset removes the workspace root setting from a
              workspace and the associated metadata.  If there are
              any modules populated, you cannot unset the workspacee
              root.

              -set sets the workspace root setting on a workspace and
              creates the initial metadata. (Default)
```

-[un]set (File-based)

```
-[un]set      Indicates whether to set the workspace root or remove
              the workspace root setting from a workspace.

              -unset removes the workspace root rsetting from a workspace
              and the associated metadata.

              -set sets the workspacee root setting on a workspace and
              creates the initial metadata. (Default)
```

--

```
--          Indicates that the command should stop looking for
            command options. Use this option when arguments
            to the command begin with a hyphen (-).
```

RETURN VALUE

```
This command returns an empty string "" on success.  If the command
fails, it returns a failure message detailing the reason for the
failure.
```

SEE ALSO

```
url root, mkmod, populate, command defaults
```

EXAMPLES

Setting the Workspace Root for a Module (Module-based)

```
This example shows setting the workspace root directory for the
MyModules workspace.

stcl> setroot MyModules
  Set Root operation successfully completed.
```

Unsetting the Workspace Root for a Module (Module-based)

```
This example shows unsetting the workspace root directory for the
MyModules workspace.

dss> setroot -unset MyModules
 There are modules present in this workspace root. They must be
 removed first.

dss> rmmod MyModules/Chip%1
...

dss> setroot -unset MyModules
 Set Root (unset) operation successfully completed.
```

Setting the Workspace Root For Files-Based Objects (File-based)

```
This example shows setting the workspace root directory for
files-based objects.

dss> setroot ./projects
 Set Root operation successfully completed.
```

setselector

setselector Command

NAME

```
setselector         - Sets the persistent selector list
```

DESCRIPTION


```
This command sets the persistent selector list, as stored in an
object's local metadata, for the specified objects. Any previous
selector list is overwritten or cleared.

Note that two commands other than 'setselector' can also update the
persistent selector list of an object:

  o The setvault command supports the following syntax:
    setvault [-recursive] <vault>@<selectorList> <workareaFolder>
    which is equivalent to doing a 'setvault' followed by
    a 'setselector'.

  o The populate command sets the persistent selector of the
    workspace when a version is specified using the -version option.

Note:
To resolve a selector, DesignSync does not search above the workspace root
of
a workspace. Thus, if the workspace root is set on a folder
(/Projects/ASIC/alu) and you apply the 'setselector' command at a
higher-level folder (for example, /Projects/ASIC), the 'setselector'
command is ignored at and below the folder where the setvault
occurred (/Projects/ASIC/alu).

For single-branch environments, you may not need the setselector
command. The default persistent selector list is 'Trunk', which is
the default branch tag for branch 1. If you will not be working
with additional branches, this default 'Trunk' selector may be
sufficient.

The 'P' data key for the 'ls' command and the 'url selector'
command report an object's persistent selector list.

To clear, or unset, the persistent selector list, specify an empty
string ("") as the selector-list argument. Clearing the persistent
selector list restores the default behavior of having an object
inherit its persistent selector list from the parent folder. Any
persistent selector list in local metadata is removed.

Note:  You cannot unset the selector list from the UNIX command line
using the  DesignSync Concurrent Shell (dssc) client because null
strings ("") are not passed from the UNIX command line to the
DesignSync client.  In order to clear the persistent selector list
without invoking a DesignSync client, you must use the Synchroncity
Tcl Shell (stcl) with the -exp option, for example:
  $ stcl -exp 'setselector "" <argument>'
```

The object arguments to the 'setselector' command can be versionable objects (files or collections), local folders, or top-level modules. The object's persistent selector list is set to the specified value. If you are doing a recursive setselector, all subfolders and objects in the hierarchy have their persistent selector lists cleared (unless a subfolder is configuration-mapped; see Configuration Mapping).

Important: Persistent selector lists set on subfolders or individual managed objects in a work area are not obeyed by the 'populate -recursive' command. Therefore, the 'setselector' command issues a warning when you set the persistent selector list on an object to a value that differs from its inherited value.

Notes for Using setselector (Module-based)

When using the ci and import commands, you can override the persistent selector on a per-operation basis with the -branch or -version options. When using the populate command with the -version tag, the persistent selector is automatically updated to match the command specified version.

Notes for Using setselector (File-based)

The following DesignSync commands use the persistent selector list to determine what version or branch to operate on: populate, ci, co, import. You can override the persistent selector list on a per-operation basis with the -version option (for populate, co, and import) or -branch option (for ci). However, using -version or -branch does not change the persistent selector list.

Valid Selectors for Module Objects (Module-based)

The selector-list argument is a comma-separated list of one or more selectors. The list cannot contain whitespace. Valid selectors are:

- Top-level modules.
  Note: Persistent selectors can only be set on a top-level module.
- Branch and version numbers:
  - 1.2.4 (A branch has an odd number of period-separated numbers.)
  - 1.2.4.1 (A version has an even number of period-separated numbers.)
- Version tags
- Branches specified as:
  - <branchtag>:<version>
  - <branchtag>:Latest
  - <branchtag>: (equivalent to <branchtag>:Latest)
- Date selectors specified as:
  - <branchtag>:Date(<date>)
  - VaultDate(<date>)

```
o Auto-branch selectors specified as:
  - Auto(<tag>)
  Note: Auto-branches cannot be specified for modules.

When a single selector is specified or set as the persistent selector
for a workspace, the selector is resolved and used for the operation.
When a selector list is specified, the last selector in the list
becomes the main selector for the workspace, and the objects matching
the specified selector are added into the workspace, replacing the
objects specified by the main selector, if needed, blending the
selectors sequentually up the selector list until the first item in
the list is processed as the last selector to draw from.  This
blended workspace, containing objects from multiple versions can be
checked in as a module snapshot, showing a specific combination of
objects.

 Note: You must specify branches explicitly in selector lists.
To do so, specify both the branch and version as follows:
'<branchtag>:<versiontag>', for example, 'Rel2:Latest'.
You can also use the shortcut, '<branchtag>:', for
example "Rel2:".  If you don't explicitly specify the
branch selector in this way, DesignSync does not resolve
the selector as a branch selector. See the "selectors"
topic for details on selector lists, including descriptions
of these selector types.

Note: If the "HrefModeChangeWithTopStaticSelector" registry key is
enabled and the populate hrefmode is set to static when the setselector
command is run, the resolved static version is set as the persistant
selector by the command. For more information about setting the
"HrefModeChangeWithTopStaticSelector" registry key, see the ENOVIA
Synchronicity DesignSync Data Manager Administrator's Guide
```

Valid Selectors for Files-Based Objects (File-based)

```
The selector-list argument is a comma-separated list of one or more
selectors. The list cannot contain whitespace. When you specify a
selector list, the command uses the first valid selector in the
list. Valid selectors are:
 o Branch and version numbers:
  - 1.2.4 (A branch has an odd number of period-separated numbers.)
  - 1.2.4.1 (A version has an even number of period-separated
             numbers.)
o Version tags
o Branches specified as:
  - <branchtag>:<version>
  - <branchtag>:Latest
  - <branchtag>:  (equivalent to <branchtag>:Latest)
o Date selectors specified as:
  - <branchtag>:Date(<date>)
  - VaultDate(<date>)
o Auto-branch selectors specified as:
  - Auto(<tag>)
  Note: Auto-branches cannot be specified for modules.
```

```
Note:
You must specify branches explicitly in selector lists.
To do so, specify both the branch and version as follows:
'<branchtag>:<versiontag>', for example, 'Rel2:Latest'.
You can also use the shortcut, '<branchtag>:', for
example "Rel2:".  If you don't explicitly specify the
branch selector in this way, DesignSync does not resolve
the selector as a branch selector. See the "selectors"
topic for details on selector lists, including descriptions
of these selector types.
```

Configuration Mapping (Legacy-based)

```
Configuration mapping is used when a configuration name does not
have the same meaning for all sections of a project. For example, a
project's Beta configuration may consist of the Gold configuration
of one section, the Rel20 configuration of another, and several
other sections whose design files are actually tagged
Beta. Configuration mapping lets you identify these different
versions of design data with one configuration name. Configuration
mapping is implemented through sync_project.txt files that reside
in vault folders and is typically set up by a project leader.

If setselector is applied to a legacy configuration-mapped folder and
the selector you specify is mapped, the persistent selector list is
set to the mapped value. For example, if the specified selector
'Trunk' maps to the 'Gold' configuration, then the persistent
selector list is set to 'Gold'. If you are doing a recursive
setselector, then all versionable objects in the hierarchy have their
persistent selector lists cleared, and the persistent selector lists
of subfolders are: - Cleared if the folder is not configuration
mapped.  - Set to the mapped value if the folder is mapped.  - Set to
the mapped value, and that mapped value propagates to any subfolders
if the folder is mapped and also references a different vault (as
identified by the REFERENCE keyword in the sync_project.txt file).

Notes: The case where a ProjectSync configuration and its
       associated DesignSync tag have the same name is not
       configuration mapping. DesignSync does not store the mapped
       value in this case.

       When populating a configuration-mapped folder, the populate
       command changes the persistent selector to the selector list
       so you do not need to use the setselector command. This
       behavior is a performance optimization so that future
       check-out operations have the configuration-map information
       locally instead of requiring additional SyncServer
       communication.
```

SYNOPSIS

```
setselector [-recursive] [-selected] [--]
            <selector>[,<selector>...] <argument> [argument>...]
```

SELECTORS

- -selector

-selector

```
<selector>      Set the persistent selector, or selector list to the
                argument.  For a full list of allowed selectors, see
                the command Description section.
```

ARGUMENTS

- Workspace Module (Module-based)
- Workspace Folder
- Workspace Objects

Workspace Module (Module-based)

```
<workspace module>  Sets the selector on the specified workspace
                    module.
```

Workspace Folder

```
<workspace folder>  Sets the selector on the specified workspace
                    folder.
```

Workspace Objects

```
<workspace object>  Sets the select on the specified workspace
                    object.  The object cannot be a member of a
                    module.
```

OPTIONS

- -recursive (Module-based)
- -recursive (Legacy-based)
- -recursive (File-based)
- -selected
- --

-recursive (Module-based)

```
 -recursive       Perform this operation on all objects in all
                  subfolders in the hierarchy. DesignSync sets the
                  selector list on the top-level folder, and clears
                  the persistent selector list for each object in
                  the hierarchy. Clearing the persistent
                  selector list restores the default behavior of
                  inheriting the persistent selector list from the
                  folder on which the setselector command was
                  applied.

                  If the setselector command reaches a static href, it
                  does not operate recursively on that submodule. It
                  also does not operate recursively into external
                  modules, legacy modules, or references to file-based
                  vault objects.
```

-recursive (Legacy-based)

```
 -recursive       Perform this operation on all objects in all
                  subfolders in the hierarchy. DesignSync sets the
                  selector list on the top-level folder, and clears
                  the persistent selector list for each object in
                  the hierarchy, unless a subfolder is
                  configuration-mapped.  For information on how
                 DesignSync behaves when a subfolder is configuration
                 mapped, see Configuration Mapping in
                  the Description section. Clearing the persistent
                  selector list restores the default behavior of
                  inheriting the persistent selector list from the
                  folder on which the setselector command was
                  applied.
```

-recursive (File-based)

```
 -recursive       Perform this operation on all objects in all
                  subfolders in the hierarchy. DesignSync sets the
                  selector list on the top-level folder, and clears
                  the persistent selector list for each object in
                  the hierarchy. Clearing the persistent selector list
                 restores the default behavior of inheriting the
                 persistent selector list from the folder on which
                 the setselector command was applied.
```

-selected

```
 -selected        Perform this operation on objects in the select
                  list (see the 'select' command) as well as the objects
```

```
                specified on the command line. If no objects are
                specified on the command line, this option is
                implied.

                Note: 'Select lists' and 'selector lists' are two
                distinct features. 'Select lists', as managed
                by the 'select' and 'unselect' commands and used
                by commands that support the '-selected' option,
                are an optional way to specify on which objects
                DesignSync commands should operate. 'Selector
                lists', as managed by the 'setselector' command
                and the '-version' and '-branch' options to
                various commands, specify on which version or
                branch of a given object DesignSync commands
                should operate.
```

--

```
 --             Indicates that the command should stop looking for
                command options. Use this option when arguments
                to the command begin with a hyphen (-).
```

RETURN VALUE

```
In dss/dssc mode, you cannot operate on return values, so the return
value is irrelevant.

In stcl/stclc mode, two lists are returned, where the first list
is non-empty if at least one object was successfully processed,
and the second list is non-empty if at least one object failed.

Notes:
 - Scripts should only test for non-empty lists to
   determine success or failure. The actual content of a non-empty
   list currently takes the form "Objects succeeded (n)" and
   "Objects failed (n)", where 'n' is the number of objects that
   succeeded or failed. However, these return values are subject to
   change in a future release.
 - If all objects fail, an exception occurs (the return value is
   thrown, not returned).
```

SEE ALSO

```
ci, co, populate, tag, selectors, setvault, url selector
```

EXAMPLES

- Example of Using Setselector with Module Snapshots (Module-based)

Example of Using Setselector with Module Snapshots (Module-based)

```
The following examples shows setselector in a blended environment that
has a module snapshot, Gold, and a main selector, Trunk:Latest.

This example shows setting the selector to the Gold snapshot with a
main selector Trunk:Latest in the Chip module.  It is not recursive
and does not affect the submodules in the module hierarchy.

   dss> setselector Gold,Trunk: Chip%0

This example removes the overlay selector and does not modify the main
selector.  It is not recursive and does not affect submodules in the
module hierarchy.

   dss> setselector Trunk: Chip%0

This example sets the selector list recursively and modifies the main
selector for the top level module only.  You cannot modify the main
selector for the submodules using the setselector command. The
command output will remind you that the main selector was not changed
within the submodules.

   dss>  setselector -rec  Gold,Trunk:Latest Chip%0
```

Example Using the Persistent Selector List in a multi-branch environment (File-based)

```
The team methodology is that 'gold', 'silver', and 'bronze' are
version tags, and 'Main' is the branch tag for the Main
branch. Therefore, for each object fetched by the populate
operation, DesignSync uses the following search order:
  1. Fetch the version tagged 'gold'.
  2. Fetch the version tagged 'silver'.
  3. Fetch the version tagged 'bronze'.
  4. Fetch the Latest version on the 'Main' branch.

The following example sets the persistent selector list for local
folder MUX1, and all objects and subfolders within it, to
'gold,silver,bronze,Main:Latest'.

  dss> setselector -recursive gold,silver,bronze,Main:Latest MUX1
  dss> scd MUX1
  dss> populate
```

Example of Using Setselector to Auto-Branch (File-based)

```
In the following example, you want to auto-branch off the 'Trunk'
branch to try 'what if' scenarios. You recursively set the
```

```
persistent selector list for local folders Mod1 and Mod2 to
'Auto(Dev),Trunk'.

  dss> setselector -recursive Auto(Dev),Trunk Mod1 Mod2

When you check in an object, DesignSync uses the Auto(Dev)
selector, which checks in the new version to the 'Dev' branch,
creating the branch if necessary. If you check out an object,
DesignSync fetches the Latest version from the 'Dev' branch if
'Dev' exists, or the Latest version from 'Trunk' otherwise.

The following example clears the persistent selector list for
the current folder and all subfolders so that the persistent
selector list is inherited from the parent folder:

  dss> setselector -rec "" .

The following example sets the persistent selector list for file
'test1.v' to 'Rel2.1:Latest'. Future checkins and checkouts of
'test1.v' will take place, by default, on the Rel2.1 branch.

  dss> setselector Rel2.1:Latest test1.v

Note that using 'setselector' on individual files is
not recommended, because inconsistent selector lists between
objects and the top-level folder are not obeyed during
populate operations.
```

setvault

setvault Command

NAME

```
setvault            - Associates a vault with a work area
```

DESCRIPTION

- Note for Module Workspaces (Module-based)
- Using setvault with Modules (Module-based)
- Using setvault with DesignSync objects (File-based)

```
This command maps a local folder (directory) to a revision-control
vault folder (repository). If there is no workspace root directory
already set above the local folder, the root directory will be
defined one level above the highest folder level containing a defined
vault connection by default or as defined on the Workspace panel in
SyncAdmin. For more information Workspace root definition, see the
DesignSync Data Manager Administrator's Guide.
```

Note: You can remove the mapping using the unsetvault command.

Note for Module Workspaces (Module-based)

You can disable automatically setting the workspace root setting for module workspaces. For more information see the DesignSync Data Manager Administrator's Guide.

This command is subject to access controls on the server. See the ENOVIA Synchronicity Access Control Guide for details.

Using setvault with Modules (Module-based)

Setvault should only be run if you have relocated a module, for example, by moving it to a different disk or server. A module relocated to a different physical location retains its unique module identifier. When setvault is run on that module, DesignSync verifies that the new vault contains a module with the same name and unique identifier as the one in the workspace before performing the setvault. If the vault does not contain a module with the same name and identifier, the command fails.

Note: You can only run setvault on a top-level module, not on one fetched by a hierarchical reference from a higher-level module.

Note: If the "HrefModeChangeWithTopStaticSelector" registry key is enabled and the populate hrefmode is set to static when the setvault command is run, the resolved static version is set as the persistant selector by the command. For more information about setting the "HrefModeChangeWithTopStaticSelector" registry key, see the ENOVIA Synchronicity DesignSync Data Manager Administrator's Guide

Using setvault with DesignSync objects (File-based)

Setting the vault is the first step in placing design data under revision control or checking out (populating) data that is already managed.

Every local folder and file has a default client vault even if you have not explicitly set the vault. Client vaults:

- Reside in the location determined during installation of your client
- Cannot be accessed by other users, so you should only use the client vault to manage private data
- Are always identified using a file: URL

You must explicitly set the vault for a folder before you can check in objects contained in the folder.

Typically, you use server (remote) vaults, which are managed by

```
Synchronicity servers (SyncServers), instead of client
vaults. Server vaults:
  - Can reside on your local host, but often reside on another host
  - Can be accessed by any user who is authorized to do so
  - Are always identified using a sync: URL

When you set the vault on a folder, that vault association is
stored in the local metadata for that folder. Each subfolder
inherits its vault association from its parent folder; the vault
association is not stored in metadata unless the subfolder has
an explicit setvault applied to it. Every versionable object (file
or collection) in the hierarchy inherits its vault from the
parent folder, although once a revision-control operation has been
performed on the object, the vault association is stored in that
object's metadata. Therefore, anytime you want to change a vault
setting (as opposed to setting the vault for the first time), use
the -recursive option. The recursive operation removes all vault
associations that are stored in metadata, which causes all objects
in the hierarchy to inherit their vault associations from the
folder on which the setvault is applied.

Notes:

o To resolve a selector, DesignSync does not search above the root of
  a workspace where a setvault has been applied. Thus, if a folder
  has no selector or persistent selector set, DesignSync searches
  up the hierarchy only as far as the first folder that has a vault
  association.

o If the number of characters in the path to the vault exceeds 1024,
  revision control operations may fail.

o Vault settings on subfolders in a work area are not obeyed
  by the 'populate -recursive' command. Consider using
  REFERENCEs in sync_project.txt files to redirect a subfolder to a
  different vault. See the Design Reuse book in DesignSync Data
  Manager User's Guide for more information.

When you use the 'setvault' command:
- The specified SyncServer must be running.
- If you specify a vault that does not exist, you get a warning so
  you can make sure that your vault path is correct. When you
  specify a new vault, the vault folder is not created until you
  check in design data.
- The local folder on which you are setting the vault must exist.
- You must have write permission for the parent folder of the
  folder for which you are setting the vault. You need write
  permission in order for DesignSync to create local metadata (as
  stored in .SYNC directories) for the parent folder.
```

SYNOPSIS

```
setvault [-recursive] [--] <vaultURL>[@<selector>[,<selector>...]]
         <localFolder>
```

ARGUMENTS

Vault URL

```
<vaultURL>     Specify the new location on the server for the
               top-level module or DesignSync object or folder.
               module should be specified in the following form:
               <protocol>://<host>:<port>/[Modules|Projects/] <path>

               o Protocol indicates whether to use a standard
                 connection or an SSL connection.  For a standard
                 connection use "sync" as the protocol.  For an SSL
                 connection, use "syncs" as the protocol.
               o Host is the machine on which the vault's SyncServer
                 is running. Specify a full domain name, such as
                 myhost.myco.com. You can specify just the machine
                 name ('myhost' in this example) if you are on the
                 same LAN as the SyncServer host machine.
               o port is the SyncServer port. You can omit the
                 port specification if the SyncServer is
                 using the default port of 2647.
               o path is the path to the vault you are creating
                 or accessing. For a client vault, the path
                 is the full, absolute path on your local
                 machine. For a server vault, the path is
                 relative to the server root as specified
                 during the SyncServer installation.

               Note: You must specify a top-level module folder.  The
               setvault command does not work on referenced modules.
```

Local Module (Module-based)

```
<localModule>  Specify the local module or folder to set as the
<localFolder>  workspace path for the server module or DesignSync
               folder.
```

Local Folder (File-based)

```
<localFolder>  Specify the local folder to set as the
               workspace path for the DesignSync folder.
```

OPTIONS

-recursive (File-based)

```
-recursive      The vault associations for all objects in the work
                area are updated so that they inherit the specified
                vault. Use -recursive when you are changing a
                vault specification (as opposed to setting the
                vault for the first time).

                CAUTION: If a subfolder had an explicit setvault
                applied to it (so that the vault information is
                stored in the folder's local metadata), that vault
                association is removed and the subfolder reverts
                to inheriting its vault association from the parent
                folder.
```

--

```
--              Indicates that the command should stop looking for
                command options. Use this option when arguments
                to the command begin with a hyphen (-).
```

RETURN VALUE

```
none
```

SEE ALSO

```
unsetvault, populate, unlock, tag, cancel, ci, co, url vault,
setselector, selectors, setroot
```

EXAMPLES

Example of Associating a Server Vault with the Current Folder

```
This example associates a server vault with the current folder.
The vault directory 'Projects/Sportster' is relative to
```

```
 the server root directory that was specified during server
 installation.
   dss> setvault sync://holzt.myco.com:2647/Projects/Sportster .
```

Example of Associating a Server Vault with a Specified Directory

```
 You can specify the local folder using relative or absolute paths,
 and you can omit the port specification because the SyncServer is
 using the default port of 2647:
   dss> setvault sync://holzt.myco.com/Projects/Sportster ../Sportster
   dss> setvault sync://holzt.myco.com/Projects/Sportster
/home/goss/Sportster
```

Example of Changing the Vault Association Recursively in a Workspace

```
 This example changes the vault association for a work area, which
 requires the -recursive option, and sets the work area persistent
 selector list to 'auto(Debug),Main:Latest':
   dss> setvault -rec \
        sync://holzt.myco.com/Projects/Sportster@auto(Debug),Main:Latest .
```

Example of Associating a Local Vault with a Specified Directory

```
 This example creates an association between the local folder
 '/home/goss/lunarLander' and the client vault
 'file:///home/goss/myVault/lunarLander'.
   dss> setvault file:///home/goss/myVault/lunarLander /home/goss/lunarLander

 Note: Client vaults cannot be shared across project teams. Only
 specify a client vault when you alone will be accessing the data.
```

unsetvault

unsetvault Command

NAME

```
 unsetvault          - Disassociates a vault from a work area
```

DESCRIPTION

```
 This command removes the mapping between a local folder or object and
 a revision-control vault or folder (repository) by removing the
 associated metadata for the object.  This metadata is stored in the
 .SYNC directory within each DesignSync workspace folder.
```

```
The primary uses for this command are to unset a vault
association incorrectly applied to a folder, or, if a folder is
copied from one workspace to another workspace, to remove the
metadata associated with the original workspace.

If you specify an entire folder, the .SYNC directory is completely
removed.  If you specify a specific object, only the metadata for
that object is removed.

The unsetvault command is not applicable to any of the following data
objects:
  o Cached objects
  o Mirrored objects

If the unsetvault is used on any of the above data objects, or used
with the -recursive option on a folder containing any of the above
objects, the command fails without removing the vault association of
any of the objects in the workspace.

Note: Unsetvault does not remove a vault setting inherited from the
parent folder.
```

SYNOPSIS

```
unsetvault [-[no]recursive] <argument> [..]
```

ARGUMENTS

- DesignSync Object
- Workspace Folder

DesignSync Object

```
<DesignSync      Specify the name of the DesignSync object to
 object>         disassociate from the server vault.
```

Workspace Folder

```
<Workspace       Specify the local workspace folder to remove the
 folder>         vault association from.
```

OPTIONS

- -[no]recursive
- --

-[no]recursive

```
-[no]recursive  Controls whether the vault associations for all
                objects in the work area are removed.

                -recursive removes the vault association for all
               objects, including subfolders, in the workspace.

                Note: The -recursive option is only valid for
                folders.

                CAUTION: If a subfolder had an explicit setvault
                applied to it (so that the vault information is
                stored in the folder's local metadata), that vault
                association is also removed.

                -norecursive removes the vault association for the
                specified object or folder, but does not traverse the
                folder structure.
```

--

```
--              Indicates that the command should stop looking for
                command options. Use this option when arguments to
                the command begin with a hyphen (-).
```

RETURN VALUE

```
If the command is successful, DesignSync returns an empty
string ("").  If the command fails, DesignSync returns an error
explaining the failure.
```

SEE ALSO

```
setvault, populate, ci, co, url vault, setselector, selectors
```

EXAMPLES

```
This example shows using unsetvault to remove the vault setting from
chip workspace directory.  Note that after the vault setting has been
removed, the directory shows the default client_vault URL, indicating
that the vault association has not been set.

stcl> url vault ./chip

 sync://srv2.ABCo.com:2647/Projects/chip
```

```
stcl> unsetvault chip

 Removing all metadata in /home/rsmith/workspaces/chip

stcl> url vault ./chip
 file:///home/rsmith/syncdata/client_vault/home/rsmith/workspaces/example

stcl>
```

Primary Revision Control

cancel

cancel Command

NAME

```
cancel              - Cancels a previous checkout operation
```

DESCRIPTION

- Notes on Using cancel with Collections
- Notes on Using Cancel with Modules (Module-based)
- Notes on Using cancel with File-Based Objects (File-based)
- Auto-Branching for File Objects and Legacy Modules Objects (File-based)

```
This command effectively performs an "un"checkout operation on the
specified locked object. This operation unlocks objects previously
locked in that work area and leaves the objects in the specified
state.

If the object was modified locally, it remains in your directory by
default.  If you specify the "-force" option, the object is
re-fetched from the server and the local modifications are
discarded.

You lock a branch by checking out an object by using the '-lock'
option with the 'co', 'populate', or 'ci' commands. Only one user can
have a lock on an object at a time. Having a lock
prohibits other users from checking in changes to that branch;
however, other users (or the same user in different work
areas) can independently lock, unlock, and check in changes to other
branches. The cancel command only cancels a checkout you have
performed. To unlock a file locked by another user, use the unlock
command.

DesignSync determines what state to leave files in your work area
```

```
after the cancel operation completes as follows:
  1. DesignSync obeys the state option (-keep, -share,
     -mirror, -reference) specified on the command line.
  2. If no state option is specified, DesignSync uses the default
     fetch state as specified by your project leader. See
     the "fetch preference" help topic for more information.
  3. If a default fetch state is not defined, the default
     behavior for 'cancel' is -keep.

Note: If the object being operated on has been designed uncachable,
cancel automatically ignores the -share and -mirror option and
performs the operation in -get mode.  For more information, see the
caching commands.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

This command supports the command defaults system.
```

Notes on Using cancel with Collections

```
If you specify a collection member as the object to be operated on,
DesignSync skips the object and warns that the object is not
versionable. If DesignSync attempts to operate on a collection member
specified implicitly (through the use of wildcards or a recursive
operation), DesignSync silently skips the object. You can change this
behavior by using the SyncAdmin "Map operations on collection members
to owner" setting. If you select this setting and DesignSync attempts
to operate on a collection member during a revision control
operation, DesignSync determines the member's owner collection and
operates on the collection as a whole.
```

Notes on Using Cancel with Modules (Module-based)

```
o Running cancel on a workspace in the module effects a cancel on
  all the objects within that module that are populated to the
  workspace

o Module branch checkouts can not be canceled.  To remove a module
  branch lock without a checkin, you must use the unlock command.

o If an object was explicitly excluded from a cancel operation by
  -filter or -exclude the command output message indicates that the
  object was "excluded by filter."

o You cannot cancel the lock on a module member that has been removed
  or moved in the workspace.
```

Notes on Using cancel with File-Based Objects (File-based)

```
If an object was explicitly excluded from a cancel operation by
-exclude (for DesignSync objects) the command output message
indicates that the object was "excluded by filter."
```

Auto-Branching for File Objects and Legacy Modules Objects (File-based)

```
You can create a new, locked branch by using 'co -lock' with a selector
and autobranching. This branch can be unlocked without creating a new
version by:

 - Using 'cancel' from the workspace where the branch was locked.
 - Using 'unlock' on the vault.
 - Using 'ci' from the workspace where the branch was locked, without
   making modifications.

In these cases, the lock is removed from the vault, the auto-created
branch is removed, and the branch tag is deleted. If the branch is
removed but still exists in the metadata of a workspace, some commands
(such as the 'url' commands and 'vhistory') will fail with "No such
version.
```

SYNOPSIS

```
cancel [-exclude <object>[,<object>...]] [-filter <string>]
 [-[no]force] [-hreffilter <string>]
 [-keep | -share | -mirror | -reference] [-modulecontext <context>]
 [-[no]selected] [-[no]retain] [-trigarg <arg>] [--] [<argument>
 [<argument>...]]
```

ARGUMENTS

- Member Module/Member Folder (Module-based)
- Workspace Module (Module-based)
- DesignSync File Object (File-based)
- DesignSync Folder (File-based)

Member Module/Member Folder (Module-based)

<module member | module folder>	Specify a module member or module folder to cancel the lock on.

Workspace Module (Module-based)

<workspace module>	Specify the module to cancel the locks in. All locks in the module held by the user initiating the cancel are removed.

	Note: This does not remove a lock on a module branch.

DesignSync File Object (File-based)

<DesignSync object>	Specify a versionable file on the server or in your workspace (local) to cancel the lock on the object. Note: If you specify a collection member as the object to be operated on, DesignSync skips the object and warns that the object is not versionable. For more information on collections are processed by the cancel command, see the Notes section of the command Description.

DesignSync Folder (File-based)

<DesignSync folder>	Specify a DesignSync folder in your workspace to cancel the lock on all objects in the folder. To unlock all objects in sub-folders of the specified folder, use the -recursive option.

OPTIONS

-exclude

-exclude <fn>	Specifies a comma-separated list of files and

directories to be excluded from the operation. Wildcards are allowed.

Do not specify paths in your arguments to -exclude. Before operating on each object (such as during a recursive cancel), DesignSync compares the object's leaf name (path stripped off) to the exclude list to see if there is a match. Because the object's path is removed, the object will not match any object in the exclude list specified with a path. For example, if you specify '-exclude bin/*.exe', you will not successfully exclude bin/foo.exe or any other *.exe file. You need to instead specify '-exclude *.exe', or '-exclude foo.exe' if you want to exclude only 'foo.exe'. The result is that you cannot exclude a specific instance of an object -- you exclude all matching objects.

In addition to objects you specify using the -exclude option, the "These objects are always excluded" field from the Tools->Options-> General->Exclude Lists dialog box in DesignSync lists objects that are always excluded from revision-control operations.

-filter (Module-based)

-filter <string>

Specify one or more extended glob-style expressions to identify an exact subset of module objects on which to operate. Use the -exclude option to filter out DesignSync objects that are not module objects.

The -filter option takes a list of expressions separated by commas, for example: -filter +top*/.../*.v,-.../a*

Prepend a '-' character to a glob-style expression to identify objects to be excluded (the default). Prepend a '+' character to a glob-style expression to identify objects to be included. Note that if the list of expressions begins with an include character ('+'), the filter excludes all objects except those that match the include string.

Specify the paths in your glob-style expressions relative to the current directory, because DesignSync matches your expressions relative to that directory. For submodules followed through hrefs, DesignSync matches your expressions against the objects' natural

paths, their full relative paths. For example, if a module Chip references a submodule, CPU, and CPU contains a file, '/libs/cpu/cdsinfo.tag', DesignSync matches against '/libs/cpu/cdsinfo.tag', rather than matching directly within the 'cpu' directory.

If your design contains symbolic links that are under revision control, DesignSync matches against the source path of the link rather than the dereferenced path. For example, if a symbolic link exists from 'tmp.txt' to 'tmp2.txt', DesignSync matches against 'tmp.txt'. Similarly for hierarchical operations, DesignSync matches against the unresolved path. If, for example, a symbolic link exists from dirA to dirB and dirB contains 'tmp.txt', DesignSync matches against 'dirA/tmp.txt'.

The extended glob-style expressions you use to filter the objects are standard glob-style expressions, but they are extended so that you can use the "..." syntax to indicate that the expression matches any number of directory levels. For example, the expression, "top/.../lib/*.v" matches *.v files in a directory path that begins with "top", followed by zero or more levels, with one of those levels containing a lib directory. The command traverses the directory structure. If a directory name matches an exclude clause of the filter, then the entire directory and all its contents are filtered (the command stops descending at that point), otherwise the command continues traversing the directory structure searching for matching objects.

The exclude list set using SyncAdmin's General=>Exclude Lists tab takes precedence over those set by -filter; the items in the exclude list are combined with the filter expression. For example, an exclude list of "*%,*.reg" combined with '-filter .../*.doc' is equivalent to '-filter .../*.doc,.../*%,.../*.reg'.

-[no]force

-[no]force

Specifies whether to overwrite locally modified objects with the server version after removing the lock on the objects.

-noforce does not remove the file if a file

has been locally modified. It does remove the lock, leaving the locally modified file in the workspace. (Default)
-force removes the lock and the file even when the object was modified locally.

Note: If a file is locally modified, and you do not specify -force in conjunction with -share or -mirror, the cancel operation fails.

-hreffilter (Module-based)

-hreffilter <string> Excludes href values during recursive operations on module hierarchies. Because hrefs link to submodules, you use -hreffilter to exclude particular submodules. Note that unlike the -filter option which lets you include and exclude items, the -hreffilter option only excludes hrefs and, thus, their corresponding submodules.

Specify the -hreffilter string as a glob-style expression. The string must represent a simple leaf name; you cannot specify a path. DesignSync matches the specified href filter against hrefs anywhere in the hierarchy. Thus, DesignSync excludes all hrefs by this leaf name; you cannot exclude a unique instance of the href.

You can prepend the '-' exclude character to your string, but it is not required. Because the -hreffilter option only supports excluding hrefs, a '+' character is interpreted as part of the glob expression.

-keep

-keep Specifies whether to keep a local copy of objects after canceling a lock operation. You can change whether the local file is read-only or read/write by default by using the "Check out read only when not locking" option from the Tools->Options->General dialog box in the graphical interface.

This option is the default object-state option unless a default object state has been defined (see the "fetch preference" help topic for more information).

Note:
- A locally modified object is left in your directory by default unless you choose "-force", in which case the object is re-fetched from the server and the local modifications are discarded.

- If you fetch an object as a locked reference (using co -lock -reference, for example), specifying 'cancel -keep' for that object cancels the lock and fetches the file. To cancel the lock and keep a reference to the file, use 'cancel -reference'.

-mirror (File-based)

-mirror

Create a symbolic link to the file in the mirror directory. This option requires that you have associated a mirror directory with your working directory with the setmirror command. In addition, the effective workspace selector (set using 'setselector', 'setvault', or the -branch option) must match the mirror workspace selector.

Note:
o This option is not supported on Windows platforms.
o When operating on a mirror directory, the cancel operation does not require an exact match between the workspace selector and the mirror selector in the case of <BranchName>: or Trunk selectors.
The cancel operation considers:
- A selector of 'Trunk' to be the same as 'Trunk:' and 'Trunk:Latest'
- A selector of <BranchName>: to be the same as <BranchName>:Latest

-modulecontext (Module-based)

-modulecontext <context>

Identifies the module on which the cancel operates. The -modulecontext option restricts the cancel operation to only a particular module if your workspace has overlapping modules.

Specify the desired module using the module name (for example, Chip), module instance name (for example, Chip%0 or /home/Modules/Chip%0).

Note that you cannot use a -modulecontext option to operate on objects from more than one module; the -modulecontext option takes only one argument, and you can use the -modulecontext option only once on a command line.

-[no]recursive (Module-based)

-[no]recursive

Determines whether to cancel the lock on the objects in the specified folder or all objects in the folder and all objects in the subfolders. This option is ignored if the argument is not a module or folder.

-norecursive removes locks only from objects in the specified folder or module. It does not remove locks from any subfolders or submodules of the specified argument. (Default)

-recursive removes locks from the specified folder and all subfolders. If the object is a module, it removes all locks from the module objects and all objects in the subfolders, and submodules.

Note: The -modulecontext option can be used to limit the operation of -recursive to only removing locked members of the specified module.

Note: On GUI clients, -recursive is the initial default.

-[no]recursive (File-based)

-[no]recursive

Determines whether to cancel the lock on the objects in the specified folder or all objects in the folder and all objects in the subfolders. This option is ignored if the argument is not a DesignSync folder.

-norecursive removes locks only from objects in the specified folder. It does not remove locks from any subfolders of the specified argument. (Default)

-recursive removes locks from the specified folder and all subfolders.

Note: On GUI clients, -recursive is the initial default.

-reference

-reference	Keep a reference to the file in the directory after the cancel operation. A reference does not have a corresponding file on the file system but does have DesignSync metadata that makes it visible to Synchronicity programs. Note: When operating on a collection object, you should not use the -reference option. When the -reference option is used on a collection, DesignSync creates a reference in the metadata for the collection object but member files are not processed and are not included in the metadata.

-[no]retain

-[no]retain	Retain the 'last modified' timestamp of the checked-out object as recorded when the object was checked into the vault. The -retain option is applicable only when the cancel operation is dealing with physical copies, as is the case when you specify the -keep option. The -share and -mirror options create links to shared objects, so timestamps cannot be set on a per-user basis. The -share and -mirror options automatically use -retain behavior; objects in the mirror/cache retain their original timestamps. However, links in your work area to the cache/mirror have timestamps of when the links were created. If you specify the -reference option, no object is created in your work area, so there is no timestamp information at all. If you do not specify '-retain' or -noretain', the cancel command follows the DesignSync registry setting for Retain last-modification timestamps. By default, this setting is not enabled; therefore, the timestamp of the local object is the time of the cancel operation. To change the default setting, your Synchronicity administrator can use the SyncAdmin tool. For information, see the SyncAdmin help.

-[no]selected

-[no]selected	Determines whether the operation is performed just on the objects specified at the command line or on objects specified at the command line and objects in the select list (see the 'select' command) -noselected cancels the locks only for objects specified on the command line. (Default) -selected cancels the locks for objects specified on the command and in the select list. Note: If no objects are specified on the command line, the -selected option is implied.

-share

-share	Keep a copy of the file in the cache directory, and create a link from the working directory to the file in the cache. Note: This option is not supported on Windows platforms. If you use 'cancel -share' on a collection object, for any collection member that is a symbolic link, DesignSync creates a symbolic link to the member object itself and not to the cache. Note: Collections existing entirely of symbolic links are not supported.

-trigarg

-trigarg <arg>	Specifies an argument to be passed from the command line to the triggers set on the cancel operation. If the argument contains whitespace, enclose the argument within double quotation marks ("") if using the dss command shell or braces ({}) if using the stcl command shell.

--

--	Indicates that the command should stop looking for command options. Use this option when an argument to the command begins with a hyphen (-).

RETURN VALUE

```
In dss/dssc mode, you cannot operate on return values, so the return
value is irrelevant.

In stcl/stclc mode, two lists are returned, where the first list
is non-empty if at least one object was successfully processed,
and the second list is non-empty if at least one object failed.

Notes:
 - Scripts should only test for non-empty lists to
   determine success or failure. The actual content of a non-empty
   list currently takes the form "Objects succeeded (n)" and
   "Objects failed (n)", where 'n' is the number of objects that
   succeeded or failed. However, these return values are subject to
   change in a future release.
 - If all objects fail, an exception occurs (the return value is
   thrown, not returned).
-  Objects reported as "excluded by filter," may have been excluded
   either with the -filter option (for modules) or with the -exclude
   option (for DesignSync objects.)
```

SEE ALSO

```
caching, co, command defaults, ci, populate, select, switchlocker,
unlock
```

EXAMPLES

```
This example cancels the checkout of all files ending in
'.v', except those whose filenames begin with 'new', leaving
links to files in the cache.
  dss> cancel -share -exclude new* *.v
```

ci

ci Command

NAME

```
ci                  - Checks in the specified objects
```

DESCRIPTION

- Versions and Branches
- Changing Checkin Comments

This command checks in the specified objects, creating a new version in each object's vault.

Note: The check-in operation requires that your work area folder be associated with a DesignSync vault location on the server. Otherwise, the operation will fail.

Usually, you need to set up the vault association only once, as the first step in placing design data under revision control or before you do an initial populate of the work area. For modules, the vault association occurs automatically during populate operations. To determine if your work area is associated with a vault, use the url vault command. For information on setting up the association, see the setvault command. For information on setting up the association with a module, see the mkmod and populate commands.

If you copied managed data into your workspace, ci detects that, and fails. To omit this check, see the "Advanced Registry Settings" topic in the DesignSync Data Manager Administrator's Guide.

Note: DesignSync requires the names of objects being checked in contain only characters that are part of the standard ASCII character set. You should also avoid the following characters, which are explicitly disallowed only for module names, to minimize confusion:
~ ! @ # $ % ^ & * ( ) , ; : | ` ' " = [ ] / \ < >
Using SyncAdmin, you can explicitly disallow any or all of these reserved characters in object and path names. For more information, see the DesignSync Administrator's Guide.

This command supports Enterprise Design Synchronization. For more information on Enterprise Design Synchronization, see How Checkin Works with Enterprise Design Synchronization below.

This command is subject to access controls on the server. See the ENOVIA Synchronicity Access Control Guide for details.

```
This command supports the command defaults system.
```

Versions and Branches

```
A version is a permanent, immutable snapshot of your design
object. Each version is assigned a unique, consecutive version
number that you can use to retrieve or otherwise identify the
object in the future.

Version numbers take the form of 1.1, 1.2, 1.3, and so on, where
the number following the period identifies the version, and the '1'
preceding the period identifies the branch (branch 1, also known as
Trunk). A branch is a line of development. Projects that require
multiple lines of development (parallel development) can define
multiple branches. Version numbers on branches other than Trunk
still take the form <branch>.<version>, where <branch> is an odd
number of period-separated numbers. For example, version 1.2.4.3
is the third version on branch 1.2.4, where 1.2.4 is the fourth
branch off version 1.2, where 1.2 is the second version on branch 1.

Because branch and version numbers are not memorable, you can apply
symbolic names, called tags, to versions and branches. Tags also
let you associate related versions of different design objects,
called configurations. See the "tag" help topic for details.
```

Changing Checkin Comments

```
The checkin comments for files checked into a vault can be modified
using the url setprop command. The <new checkin comment> is optional
on the command line and the user is prompted for it if not specified.
Here is the syntax:
  url setprop <versionURL> log [<new checkin comment>]

If the user changing the comment is not the author of the version,
a note with the user name and date and stating that the comment was
changed is prepended to the new comment.
For example:
  stcl> url setprop [url vault new_d]1.5 log "New comment set from \
  other user"
  New comment set from other user
  stcl> url getprop [url vault new_d]1.5 log
  Comment changed by JerryL Jun 02 2005, 08:19:13 EDT
  New comment set from other user
  stcl>

If a comment is not specified at the command line, and DesignSync is
set to use a file editor for comments, the designated file editor is
launched, otherwise a comment can be entered interactively on the
command line.  For more information on defining a file editor for
comments, see the DesignSync Administrator's Guide, "General
Options."
```

Note: The client-side minimum comment length is checked.

Understanding the Output

The ci command provides the option to specify the level of information the command outputs during processing. The -report option allows you to specify what type of information is displayed:

If you run the command with the -report brief option, the ci command outputs the following information:

- Failure messages.
- Warning messages.
- Informational messages concerning the status of the checkin operation.
- Success/failure/skip status.

If you do not specify a value, or the command with the -normal option, the ci command outputs all the information presented with -report brief and the additional information for each successful object checkin, excluded objects, or omitted objects.

If you run the command with the -report verbose option, the ci command outputs all the information presented with -report normal and information about each object examined or filtered.

If you run the command with the -report error option, the ci command outputs the following information:

- Failure messages.
- Warning messages.
- Success/failure/skip status messages.

Object States (Module-based)

DesignSync determines what state to leave objects checked into your work area as follows:

1. DesignSync obeys the state option (-keep, -lock, -share, -reference) specified on the command line.
2. If no state option is specified, DesignSync uses the default fetch state as specified by your project leader. See the "fetch preference" help topic for more information.
3. If no default fetch state is defined, the default behavior for ci is -keep.

Important:

- The ci command processes only locked or modified objects. DesignSync changes the state of only those objects that have been checked in. To set all of the objects in your work area to the same state, use 'populate -unifystate'. To check in unmanaged (new) objects, use 'ci -new'.

- PreFolder check-in triggers fire on folders that contain locked or modified objects being checked in. For some check-in operations like recursive checkins with -force or -new options, preFolder triggers might also fire on folders that do not contain modified or locked objects being checked in.

- The fetch state of moved module members does not change during checkin unless content of the object, indicated by a modification to the timestamp, has changed.

- If the object is designated as uncachable, attempts to place objects in the cache (ci -mirror; ci -share) will automatically populates the workspace with unlocked copies (-keep mode). For more information on cachability, see the "caching" commands.

By default (unless you use the -force option), DesignSync does not create a new version when you attempt to check in an object that is not locally modified. An object is defined as "locally modified" if its timestamp has been changed or it is a module member that has been moved with the mvmember command with the -noimmediate option.

For collections that have local versions, the check-in operation usually does not change the set of local versions in your workspace. However, there is an exception to this behavior. The check-in operation changes the set of local versions in your workspace when the originally fetched state of the object was Cache or Mirror. In this case, the check-in operation replaces files linked to the cache or mirror with physical copies.

Determining the Objects to be Checked In (Module-based)

By default, DesignSync only checks in modified objects. An object is considered modified when it meets the following criteria:

* The current timestamp of the object in the workspace is later than the fetched timestamp of the object.
* The current size or checksum of the object is different than the fetched object size or checksum.
* The module member is in the added/moved/removed state.

Note: If a hierarchical reference to a submodule version has been overridden by a higher-level href, the hierarchical reference within the parent module is NOT considered modified.

Determining Which Branch is Selected for the Check In (Module-based)

Arguments to the ci command must represent versionable objects (modules, module members, or collections), or local folders (only meaningful when you use the -recursive option). The ci command operates on the current or specified branch of each of these objects. When you are in a multibranch design environment, DesignSync determines what branch you want to check into as follows:

1. DesignSync obeys the -branch option, operating on the Latest version on the specified branch. Using this option is not typical, however, because the default behavior (without -branch) is usually the correct and intuitive behavior.

2. If -branch is not specified and you have the current branch locked in your work area, DesignSync checks into the current branch.

3. Otherwise, DesignSync uses the first selector of the object's persistent selector list ('Trunk' by default, or as defined by the setselector command). If the selector does not resolve to 'Trunk' or some other valid branch for the object (specified as <branch>:<version>, for example Rel2:Latest), the operation fails.

   Complex selector lists are a powerful capability for populate and check-out operations, but they can be dangerous for check-in operations. When creating a new version, there should be no uncertainty as to which branch to create the version on. Therefore, ci considers only the first selector in the persistent selector list.

See the "selectors" help topic for details on selectors, selector lists, and persistent selector lists.

For more details about checking in modules and module objects, see "Checking In Module Objects" below.

Filtering or Excluding Objects From Checkin (Module-based)

DesignSync features three ways to control the objects being checked in. For objects in source control, exclude and filter lists are used to exclude/include DesignSync objects. Filter lists are used to include or exclude module objects or to include DesignSync objects. Exclude lists are used to exclude DesignSync objects. Both Filter and Exclusion lists can be saved with command defaults or specified using the -fiter or -exclude option.

Note: Regardless of whether -filter or -exclude is used to exclude an object, the command output message indicates that the object was "excluded by filter." For more information on filters, see the -filter and -exclude options in the options section.

For objects that are not in source control, exclude files can be created on a per directory basis to prevent unmanaged objects from being checked in. Objects that are excluded by exclude files cannot be reincluded by a filter. Exclude files are processed before the filter and exclude options set either by the command defaults or specified on the command line. For more information on setting up exclude files, see the ENOVIA Synchronicity DesignSync Data Manager Administrator's Guide.

If you specify a collection member as the object to be operated on, DesignSync skips the object and warns that the object is not versionable. If DesignSync attempts to operate on a collection member specified implicitly (through the use of wildcards or a recursive operation), DesignSync silently skips the object. You can change this behavior by using the SyncAdmin "Map operations on collection members to owner" setting. If you select this setting and DesignSync attempts to operate on a collection member during a revision control operation, DesignSync determines the member's owner collection and operates on the collection as a whole.

Checking in Module Objects (Module-based)

The ci command recognizes and checks in modules, and their members. A module is data that represents a level of a design hierarchy. Such data includes objects or an entire vault folder hierarchy of objects managed in DesignSync, as well as hierarchical references to other modules. These modules can be stored on other SyncServers. For more information about modules, see DesignSync Data Manager User's Guide: "What is a Module?".

The ci operation checks in modified objects. This can include not only objects with modified content, but also added, moved, renamed, or removed module members. The add command always operates in no immediate mode, meaning that any objects in an Added state are added when the next checkin operation affecting the Added module members occurs. The mvmember and remove commands operate by default in a "noimmediate" mode. If you added an object to a module using add, the object is considered to be managed already; in this case, use ci without the -new option to check in the object.

When working with module data, the module object is version-controlled; module members are not independently version-controlled. For more information about module versions, see the populate command description subtopic, "Module Version Updating". See also "Automerging of Module Objects" below. You can branch a module during check-in, for more information, see the "Module Branching" section.

Notes:

- If a non-explicitly added folder becomes empty as the result of the checkin of removed module members, the folder is removed as well. If an explicitly added folder is moved, but the full contents of folder are not, the explicitly added folder remains in the same position with the unmoved contents and a new implicitly added folder is created to contain the moved contents.

- Moved module members with no content changes are moved, but the module member version is not incremented and the keywords within the file are not updated.

- If there are no content changes to the module members, the objects retain the same state in the workspace. For example, if the objects are fetched in -get mode, a file is renamed but not otherwise

modified, and then the checkin is done in -share mode, the renamed file remains in -get mode.

A module checkin is an atomic operation; if a failure occurs during a module check-in, the ci command does not check in any of the specified module objects. After you resolve the failure, you can re-apply the ci command. By default, DesignSync optimizes the check-in by continuing where the failed check-in left off. Specify the -noresume option to start the check-in from scratch.

Note: If there are structural changes to the module, such as removed or moved module members, the checkin always defaults to -noresume. The -resume operation is not applicable to module checkin operations with the -branch option.

You can use ci to check in entire modules or their members as follows:

o To check in a single module without checking in its submodules, specify the workspace module and apply the ci command without the -recursive option.

  The command checks in the module members without following hierarchical references (hrefs).

o To check in all objects in an entire module hierarchy, specify the workspace module and use the ci command with the -recursive option.

  The command traverses the hierarchy in a module-centric fashion, checking in all of the objects in the module and following its hrefs to check in its referenced submodules.

  Notes:
    The ci command does not traverse legacy modules, even if you specify the -recursive option for the module. You must check in the legacy sub-module separately.

    If you specify ci with -new and -modulecontext is selected or smart module detection is able to identify the target module, unmanaged files are checked into the appropriate target module. When ci -new -recursive is specified, the operation does not traverse hierarchical references. The ci -new -recursive operation does traverse the hierarchy in a folder-centric method and smart module detection appropriately identifies new members are belonging to the appropriate module or sub-module. For more information on how smart module detection determines the target module, see the ENOVIA Synchronicity DesignSync Data Manager User's Guide topic: Understanding Smart Module Detection.

o To check in all modified objects in a folder and its subfolders, specify a folder name and apply the ci command with the -recursive option.

  The command traverses the folders in a folder-centric fashion, checking in the modified objects in the folder and its subfolders, but without following hrefs. To follow hrefs, you must specify a

workspace module instead of a folder.

Note: Smart module detection for new module members always works in a folder-centric, not a module-centric fashion.

o To check in new files to a module, you should add the files with add, and then check in the files normally. The ci command with the -new option only checks in new files, when smart module detection can detect the target module or when the -modulecontext option specifies the module for the objects (ci -new -modulecontext <context>)

o To check in files to a new branch, specify the module context and the branch options. The -new and -recursive options cannot be used to check into a new branch. For more information on module branching, see the Branching Modules section.

Notes:

o Mirrors are not supported with module objects; ci ignores the -mirror option if you use it while checking in a module object.

o If a module contains an empty folder, DesignSync checks in the empty folder.

o If the -modulecontext option is not specified when checking in a module member with the -new option, DesignSync uses smart module detection to identify the desired module. If DesignSync cannot identify the module, the command returns an error stating that the module can not be identified and recommending the use of the -modulecontext option.

Branching Modules (Module-based)

You can check in a module to a new module branch with the checkin operation. The operation creates a new module version on the branch containing all managed objects in the workspace and on the server belonging to the specified module. This includes any of the following objects:

* Added objects that have not been checked in yet.
* Modified objects belonging to the specified module.
* Unmodified objects belonging to the specified module.
* Objects that are part of the module on the server, but have not been populated into the workspace.
* Objects in the workspace that were removed on the server in a later module version.

Note: The module member version in the workspace is always considered the desired version for the ci -branch operation. If you have older member versions in the workspace, those will become the Latest version on the new branch.

When you check a module into the new branch, DesignSync automatically

modifies the workspace selector to the Latest version of the new branch tag (<Branch>:Latest).

Note: DesignSync creates an initial, empty, module version, then creates a second version containing the module member files.

The option to check into a branch requires that you check into a new branch.

You must specify a single module for checkin. You cannot recurse through hierarchical references to branch submodules.

If you have unmanaged files in the workspace that you want to include in the module checkin, add the files to the module first, then perform the checkin. You cannot specify the -new option with a module checkin to a new branch.

Automerging of Module Objects (Module-based)

As you make changes to module objects, other team members might make changes to other module objects, thus creating new versions of the module. If you then check in your module objects, object versions in your workspace no longer match the target branch. If you had been working with non-module objects, you could either merge your changes first, or specify the -skip option to force a check-in. However, for module objects, DesignSync lets you check in the objects without specifying the -skip option. In this case, DesignSync performs an 'auto-merge' of the module objects. An auto-merge merges the module changes from your workspace into the latest version of the module in the vault, to create a new module version. This auto-merge occurs at the file level; DesignSync does not attempt to merge the contents of your module objects.

During an auto-merge, DesignSync does not automatically refresh your workspace to bring in your team members' updated module objects. Consequently, an showstatus of the workspace module shows that the fetched version of the module is not the latest version. Perform a populate operation on the module to ensure that you have the latest versions of all of the module's objects.

Note that if you attempt to check in a module object and a teammate has created a newer version of that object, DesignSync does not attempt an auto-merge of that object. In this case, you must explicitly merge these objects using 'populate -merge'. See DesignSync Data Manager User's Guide to learn more about merging modules.

How Checkin Works with Enterprise Design Synchronization (Module-based)

Operations submitted with checkin that can affect the global enterprise design such as tagging and hierarchical references changes,

are stored in a queue until they are pushed to the Enterprise server.

Not all checkin operations are sent to the queue, only the ones that include global changes, such as a checkin following a non-immediate remove of hierarchical references, or a checkin with tag operation (ci -tag).

For more information on Enterprise Design management, see the Enterprise Design Administration User's Guide.

Checking in Legacy Module Data (Legacy-based)

When ci is used with legacy modules, it never traverses the module hierarchy; it will always run in a non-recursive fashion, mimicking the behavior of the former -nomodulerecursive behavior.

If -recursive is specified during a legacy module checkin, the recursive option is ignored.

Referenced legacy sub-modules are not included in the parent module checkin. Legacy sub-modules are always checked in individually.

Object States (File-based)

DesignSync determines what state to leave objects checked into your work area as follows:

1. DesignSync obeys the state option (-keep, -lock, -share, -mirror, -reference) specified on the command line.
2. If no state option is specified, DesignSync uses the default fetch state as specified by your project leader. See the "fetch preference" help topic for more information.
3. If no default fetch state is defined, the default behavior for ci is -keep.

Important:

- The ci command processes only locked or modified objects. DesignSync changes the state of only those objects that have been checked in. To set all of the objects in your work area to the same state, use 'populate -unifystate'. To check in unmanaged (new) objects, use 'ci -new'.

- PreFolder check-in triggers fire on folders that contain locked or modified objects being checked in. For some check-in operations like recursive checkins with -force or -new options, preFolder triggers might also fire on folders that do not contain modified or locked objects being checked in.

- If the object is designated as uncachable, attempts to place objects in the cache (ci -mirror; ci -share) will automatically populates the workspace with unlocked copies (-keep mode). For more information on cachability, see the "caching" commands.

By default (unless you use the -force option), DesignSync does not create a new version when you attempt to check in an object that is not locally modified. An object is defined as "locally modified" if its timestamp has been changed.

For collections that have local versions, the check-in operation usually does not change the set of local versions in your workspace. However, there is an exception to this behavior. The check-in operation changes the set of local versions in your workspace when the originally fetched state of the object was Cache or Mirror. In this case, the check-in operation replaces files linked to the cache or mirror with physical copies.

Determining the Objects to be Checked In (File-based)

By default, DesignSync only checks in modified objects. An object is considered modified when it meets the following criteria:

* The current timestamp of the object in the workspace is later than the fetched timestamp of the object.
* The current size or checksum of the object is different than the fetched object size or checksum.

Determining Which Branch is Selected for the Check In (File-based)

Arguments to the ci command must represent versionable objects (files or collections), or local folders (only meaningful when you use the -recursive option). The ci command operates on the current or specified branch of each of these objects. When you are in a multi-branch design environment, DesignSync determines what branch you want to check into as follows:

1. DesignSync obeys the -branch option, operating on the Latest version on the specified branch. Using this option is not typical, however, because the default behavior (without -branch) is usually the correct and intuitive behavior.

2. If -branch is not specified and you have the current branch locked in your work area, DesignSync checks into the current branch.

3. Otherwise, DesignSync uses the first selector of the object's persistent selector list ('Trunk' by default, or as defined by the setselector command). If the selector does not resolve to 'Trunk' or some other valid branch for the object (specified as <branch>:<version>, for example Rel2:Latest), the operation fails.

   Complex selector lists are a powerful capability for populate and check-out operations, but they can be dangerous for check-in operations. When creating a new version, there should

```
     be no uncertainty as to which branch to create the version
     on. Therefore, ci considers only the first selector in the
     persistent selector list.

See the "selectors" help topic for details on selectors,
selector lists, and persistent selector lists.
```

Filtering or Excluding Objects From Checkin (File-based)

```
DesignSync features two ways to control the objects being checked
in.  For objects in source control, exclude lists are used
to exclude DesignSync objects. Exclusion lists can be saved with
command defaults or specified using the -exclude option.

For objects that are not in source control, exclude files can be
created on a per directory basis to prevent unmanaged objects from
being checked in.  Objects that are excluded by exclude files cannot
be reincluded by a filter. Exclude files are processed before the
filter and exclude options set either by the command defaults or
specified on the command line.

If you specify a collection member as the object to be operated on,
DesignSync skips the object and warns that the object is not
versionable. If DesignSync attempts to operate on a collection member
specified implicitly (through the use of wildcards or a recursive
operation), DesignSync silently skips the object. You can change this
behavior by using the SyncAdmin "Map operations on collection members
to owner" setting. If you select this setting and DesignSync attempts
to operate on a collection member during a revision control
operation, DesignSync determines the member's owner collection and
operates on the collection as a whole.
```

Interaction with Mirrors (File-based)

```
When using a mirror (associated with a workspace with the
setmirror command), objects are written directly to the mirror
directory.  You must have write privileges to the mirror directory.
For the mirror write-through to occur, the effective workspace
selector (set using setselector, setvault, or the -branch
option) must also match the mirror workspace selector.  You can
disable this behavior using SyncAdmin->Site Options->Mirror
Write-through.

Notes:

o When operating on a mirror directory, the check-in operation
  does not require an exact match between the workspace selector
  and the mirror selector in the case of <BranchName>: or Trunk
  selectors.  The check-in operation considers:

  - A selector of 'Trunk' to be the same as 'Trunk:' and
   'Trunk:Latest'
```

- A selector of <BranchName>: to be the same as <BranchName>:Latest

SYNOPSIS

```
ci [-autohrefversions | -[no]hrefversions]
   [-branch <branch> | -branch auto(<branch>)]
   [-[no]comment"<text>" | -cfile <file>] [-datatype ascii | binary]
   [-[no]dryrun] [-exclude <object>[,<object>...]]
   [-filter <string>] [-[no]force] [-hreffilter <string>]
   [-[no]iflock] [-keep [-keys <mode>] | -lock
   [-keys <mode>] | -share | -mirror | -reference]
   [-modulecontext <context>] [-[no]new] [-[no]recursive]
   [-report {error | brief | normal | verbose}] [-[no]resume]
   [-[no]retain] [-[no]retry] [-[no]selected] [-[no]skip]
   [-tag <tagname>] [-trigarg <arg>] [--] [<argument> [<argument>...]]
```

ARGUMENTS

- Module Folder (Module-based)
- Module Member (Module-based)
- Workspace Module (Module-based)
- DesignSync File Objects (File-based)
- DesignSync Folder Objects (File-based)

Specify one or more of the following arguments:

Module Folder (Module-based)

<module folder>	The ci command does not check in module folders, but checks in their contents if you specify the -recursive option. If the folder contains objects that have not yet been checked in, but have been added to the module using the add command, you do not need to apply the -new option. If you have new items to add to the workspace, you can use the -new option and smart module detection will determine the target module for the candidate member, or you can explicitly specify the module with the -modulecontext option.

Module Member (Module-based)

<module member>	Checks in the module member. If the member has not yet been checked in, but it has been added to the module using the add command, you do not

need to apply the -new option. If you have new items to add to the workspace, you can use the -new option and smart module detection will determine the target module for the candidate member, or you can explicitly specify the module with the -modulecontext option.

Workspace Module (Module-based)

<workspace module> Checks in the workspace module, creating a new version of the module. The check-in process checks in each updated member, but also registers other changes made to the module since the last check-in, such as versions of referenced submodules.

Note: If you are trying to do a hierarchically recursive checkin (-recursive), you can't checkin new items that have not already been added. For more information, see the -new and -recursive options.

DesignSync File Objects (File-based)

<DesignSync object> Checks the object into its vault. If the object is unmanaged, apply the -new option.

DesignSync Folder Objects (File-based)

<DesignSync folder> The ci command does not check in local folders, but checks in their contents if you specify the -recursive option. If the folder contains unmanaged objects, apply the -new option.

OPTIONS

-autohrefversions (Module-based)

```
 -autohrefversions   Processes the static hrefs based on the type of
                     checkin performed.  If the checkin is performed
                     on a module and -recursive is selected,
                     DesignSync captures the currently populated
                     versions of the module's hierarchically
                     referenced sub-modules, and records those as
                     part of the next module version, updating the
                     static hierarchical references. If the checkin
                     is performed on a file or folder within a module
                     or a module is specified, but the -recursive
                     option is not, the selected module members are
                     checked in, but the hierarchical references are
                     not updated. (Default)

                     This option is mutually exclusive with
                     -hrefversions.
```

-branch (Module-based)

```
-branch <branch>
| -branch
  auto(<branch>)
```

Performs the checkin on the branch specified by the branch or version tag, auto-branch selector, or branch numeric. This option overrides the object's persistent selector list. If a version is retrieved in the workspace, this is used as the branch-point version for any new branch created.

For a checkin using an auto-branch selector, for example Auto(Golden), if there already exists a version 'Golden', the checkin fails. However, if 'Golden' exists as a branch, the effective selector is 'Golden:Latest'; the checkin succeeds and no new branch is needed. If there is neither a version nor a branch named 'Golden' for the object, a new branch is created and it is named 'Golden'. If a version is retrieved in the workspace, this is used as the branch-point version for the new branch created. For example, if version 'Golden' is retrieved in the workspace, it is used as the branch-point version. If version 'Golden' is retrieved but it has no metadata information as a consequence of being removed from the vault earlier, DesignSync uses the latest version on branch '1' as the branch-point version. Finally, if there is no vault (in this case, the -new option must be specified), DesignSync creates a new vault (branch 1). Branch 1 is named 'Golden'.

When branching a module, you must create a new branch. You cannot specify an existing branch. The -branch tag when specified with a module is mutually exclusive with -recursive and -new. For more information on module branching, see the "Branching Modules" section in the ci command description.

Notes:

- The -branch option accepts a branch tag, a version tag, a single auto-branch selector tag, or a branch numeric. It does not accept a selector or selector list.
- The ci command ignores this option if you specify a folder as the argument and the folder contains a module object; in this case, the checkin occurs on the fetched branch. If you specify a module as the argument and use the -branch option, the checkin fails.
- The persistent selector list of the object you are checking in is not updated by the

check-in operation. Subsequent operations that use the persistent selector list will not follow the branch you just checked into. If you want to continue working on this branch, you must set the persistent selector list with the setselector command.

-branch (File-based)

-branch <branch> | -branch auto(<branch>)	Performs the checkin on the branch specified by the branch or version tag, auto-branch selector, or branch numeric. This option overrides the object's persistent selector list. If a version is retrieved in the workspace, this is used as the branch-point version for any new branch created. For a checkin using an auto-branch selector, for example Auto(Golden), if there already exists a version 'Golden', the checkin fails. However, if 'Golden' exists as a branch, the effective selector is 'Golden:Latest'; the checkin succeeds and no new branch is needed. If there is neither a version nor a branch named 'Golden' for the object, a new branch is created and it is named 'Golden'. If a version is retrieved in the workspace, this is used as the branch-point version for the new branch created. For example, if version 'Golden' is retrieved in the workspace, it is used as the branch-point version. If version 'Golden' is retrieved but it has no metadata information as a consequence of being removed from the vault earlier, DesignSync uses the latest version on branch '1' as the branch-point version. Finally, if there is no vault (in this case, the -new option must be specified), DesignSync creates a new vault (branch 1). Branch 1 is named 'Golden'. Notes: - The -branch option accepts a branch tag, a version tag, a single auto-branch selector tag, or a branch numeric. It does not accept a selector or selector list. - The -skip option is required when you are checking into a branch other than the current branch unless your current version is the branch-point version and there are no versions on the branch. For example, if you have version 1.4, you can only create versions 1.5, 1.4.1.1, 1.4.2.1, and so on, unless you use -skip. - The persistent selector list of the object you are checking in is not updated by the check-in operation. Subsequent operations

that use the persistent selector list will not follow the branch you just checked into. If you want to continue working on this branch, you must set the persistent selector list with the setselector command.

-[no]comment (Module-based)

-[no]comment "<text>"	Specifies whether to check in the specified object with or without a description of changes. If you specify -comment, enclose the description in double quotes if it contains spaces. The check-in comment is appended to the check-out comment if one was specified. The comments associated with a version are also called the "log". The ampersand (&) and equal (=) characters are replaced by the underscore (_) character in revision control notes. If you do not specify -comment, -nocomment, or -cfile DesignSync prompts you to enter a check-in comment either on the command or by spawning the defined file editor. The -cfile option is mutually exclusive with -[no]comment. For more information on defining a file editor, see the DesignSync Data Manager Administrator's Guide, "General Options." Note: If the -tag option is specified along with the -comment option, the comment text is used as both the tag comment and the checkin comment.

-[no]comment (File-based)

-[no]comment "<text>"	Specifies whether to check in the specified object with or without a description of changes. If you specify -comment, enclose the description in double quotes if it contains spaces. The check-in comment is appended to the check-out comment if one was specified. The comments associated with a version are also called the "log". The ampersand (&) and equal (=) characters are replaced by the underscore (_) character in revision control notes. If you specify -nocomment, and the object was checked out with comments ('co -comment'), the check-out comments are retained during check-in. If you do not specify -comment, -nocomment, or -cfile, DesignSync prompts you to enter a check-in comment either on the command or by

spawning the defined file editor. The -cfile option is mutually exclusive with -[no]comment. For more information on defining a file editor, see the DesignSync Data Manager Administrator's Guide, "General Options."

Note: If the -tag option is specified along with the -comment option, the comment text is used as both the tag comment and the checkin comment.

-cfile

-cfile <file>	Specifies a file containing a text comment to use as the description of the new release. DesignSync accepts a comment of any length up to 1MB. Long comments may be truncated in the output of commands that show comments. If the comment includes ampersand (&) or equal (=) characters, they are replaced by the underscore (_) character in revision control notes. This option respects the minimum comment length. The -cfile option is mutually exclusive with -[no]comment. If you do not specify one of the three options, -comment, -cfile, or -nocomment, DesignSync prompts you to enter a check-in comment either on the command line or by spawning the defined file editor. For more information on defining a file editor, see the DesignSync Data Manager Administrator's Guide, "General Options."

-datatype (Module-based)

-datatype ascii| binary	Indicates whether to disable the autodetect feature of DesignSync and create the object being checked in with the specified data type. The datatype can be changed during any module version checkin. -datatype ascii creates the new object with a data type of ascii. -datatype binary creates the new object with a data type of binary. Binary objects cannot be merged, they can only be replaced. ZIP vaults are always checked in using binary mode, regardless of whether the vault's data type is designated as ascii.

-datatype (File-based)

-datatype ascii| binary	Indicates whether to disable the autodetect feature of DesignSync and create the object being checked in with the specified data type. The datatype option can only be specified when the object is initially created. -datatype ascii creates the new object with a data type of ascii. -datatype binary creates the new object with a data type of binary. Binary objects cannot be merged, they can only be replaced. ZIP vaults are always checked in using binary mode, regardless of whether the vault's data type is designated as ascii. Note: For vault objects, this option only applies for when checking in new data. To change the data type of an existing object, use the url setprop command.

-[no]dryrun

-[no]dryrun	Specifies whether to treat the operation as a trial run; if -dryrun is specified, no objects are actually checked in. By default (-nodryrun), ci performs a standard checkin. The -dryrun option helps detect problems that might prevent the checkin from succeeding. Because local object and vault states are not changed, a successful dry run does not guarantee a successful checkin. Errors that can be detected without state changes, such as a vault or branch not existing, merge conflicts, or a branch being locked by another user are reported. Errors such as permissions or access rights violations are not reported by a dry run. Note that a dry run checkin is significantly faster than a normal checkin.

-exclude (Module-based)

-exclude <objects>	Specifies a comma-separated list of objects (collections, folders, or module objects) to be excluded from the operation. Wildcards are allowed. Note: Use the -filter option to filter module objects. You can use the -exclude option, but the -filter option lets you include and exclude

module objects. If you use both the -filter and -exclude options, the strings specified using -exclude take precedence.

Do not specify paths in your arguments to -exclude. Before operating on each object (such as during a recursive checkin), DesignSync compares the object's leaf name (with the path stripped off) to the exclude list to see if there is a match. Because the object's path is removed, the object will not match any object in the exclude list specified with a path. For example, if you specify '-exclude bin/*.exe', you will not successfully exclude bin/foo.exe or any other *.exe file. You need to instead specify '-exclude *.exe', or '-exclude foo.exe' if you want to exclude only 'foo.exe'. The result is that you cannot exclude a specific instance of an object -- you exclude all matching objects.

In addition to objects you specify using the -exclude option, the "These objects are always excluded" field from the Tools->Options-> General->Exclude Lists dialog box in DesignSync lists objects that are always excluded from revision-control operations.

-exclude (File-based)

-exclude <objects> Specifies a comma-separated list of objects (files, collections, or folders) to be excluded from the operation. Wildcards are allowed.

Do not specify paths in your arguments to -exclude. Before operating on each object (such as during a recursive checkin), DesignSync compares the object's leaf name (with the path stripped off) to the exclude list to see if there is a match. Because the object's path is removed, the object will not match any object in the exclude list specified with a path. For example, if you specify '-exclude bin/*.exe', you will not successfully exclude bin/foo.exe or any other *.exe file. You need to instead specify '-exclude *.exe', or '-exclude foo.exe' if you want to exclude only 'foo.exe'. The result is that you cannot exclude a specific instance of an object -- you exclude all matching objects.

In addition to objects you specify using the -exclude option, the "These objects are always excluded" field from the Tools->Options-> General->Exclude Lists dialog box in DesignSync lists objects that are always

excluded from revision-control operations.

-filter (Module-based)

-filter <string>

Specify one or more extended glob-style expressions to identify an exact subset of module objects on which to operate. Use the -exclude option to filter out DesignSync objects that are not module objects.

The -filter option takes a list of expressions separated by commas, for example:

```
-filter +top*/.../*.v,-.../a*
```

Prepend a '-' character to a glob-style expression to identify objects to be excluded (the default). Prepend a '+' character to a glob-style expression to identify objects to be included. Note that if the list of expressions begins with an include character ('+'), the filter excludes all objects except those that match the include string.

Specify the paths in your glob-style expressions relative to the current directory, because DesignSync matches your expressions relative to that directory. For submodules followed through hrefs, DesignSync matches your expressions against the objects' natural paths, their full relative paths. For example, if a module Chip references a submodule, CPU, and CPU contains a file, '/libs/cpu/cdsinfo.tag', DesignSync matches against '/libs/cpu/cdsinfo.tag', rather than matching directly within the 'cpu' directory.

If your design contains symbolic links that are under revision control, DesignSync matches against the source path of the link rather than the dereferenced path. For example, if a symbolic link exists from 'tmp.txt' to 'tmp2.txt', DesignSync matches against 'tmp.txt'. Similarly for hierarchical operations, DesignSync matches against the unresolved path. If, for example, a symbolic link exists from dirA to dirB and dirB contains 'tmp.txt', DesignSync matches against 'dirA/tmp.txt'.

The extended glob-style expressions you use to filter the objects are standard glob-style expressions, but they are extended so that you can use the "..." syntax to indicate that the expression matches any number of directory levels. For example, the expression, "top/.../lib/*.v" matches *.v files in a

directory path that begins with "top", followed by zero or more levels, with one of those levels containing a lib directory. The command traverses the directory structure. If a directory name matches an exclude clause of the filter, then the entire directory and all its contents are filtered (the command stops descending at that point), otherwise the command continues traversing the directory structure searching for matching objects.

The exclude list set using SyncAdmin's General=>Exclude Lists tab take precedence over those set by -filter; the items in the exclude list are combined with the filter expression. For example, an exclude list of "*%,*.reg" combined with '-filter .../*.doc' is equivalent to '-filter .../*.doc,.../*%,.../*.reg'.

-[no]force

-[no]force	Specifies whether to force the creation of a new version even if it is identical to the previous version. By default (-noforce), the timestamp of the file in the workspace is compared with the timestamp of the version in the vault. If the timestamp of the file to be checked in has not changed, then no new version is created. You might use -force to synchronize version numbers across several objects. Note that you must have a local copy of the object in your work area for a new version to be created. A new version is not created if the object does not exist or is a reference. Note: Use the -force option only if necessary. Using the -force option slows the check-in process because ci must process all objects and not just the locked or modified objects.

-hreffilter (Module-based)

-hreffilter <string>	Excludes href values during recursive operations on module hierarchies. Because hrefs link to submodules, you use -hreffilter to exclude particular submodules. Note that unlike the -filter option which lets you include and exclude items, the -hreffilter option only excludes hrefs and, thus, their corresponding submodules.

Specify the -hreffilter string as a glob-style expression. The string must represent a simple leaf name; you cannot specify a path. DesignSync matches the specified href filter against hrefs anywhere in the hierarchy. Thus, DesignSync excludes all hrefs by this leaf name; you cannot exclude a unique instance of the href.

You can prepend the '-' exclude character to your string, but it is not required. Because the -hreffilter option only supports excluding hrefs, a '+' character is interpreted as part of the glob expression.

-[no]hrefversions (Module-based)

-[no]hrefversions

Controls whether the static version of a hierarchical reference is updated.

-hrefversions updates the static version of a hierarchical reference so that the version reflects the fetched version of the corresponding submodule in the workspace.

-nohrefversions saves only the module members and does not update the hierarchical references in any way. This is particularly useful if you have not made any submodule changes locally.

The ci command ignores the -[no]hrefversion option if you are checking in non-module objects.

Note: If you check in a module using the -hrefversions option and you have checked in an updated submodule from the same workspace, the static version updates to reflect the updated submodule, rather than the fetched version.

If either of these options are specified, they override the default, -autohrefversions.

This option is mutually exclusive with -autohrefversions.

-[no]iflock (Module-based)

-[no]iflock

Specifies whether to check in all modified objects in the checkin selection or only the ones that meet any one the following criteria:
* module member is added.
* module member is removed.

* module member is moved.

-The noiflock option (Default) searches the entire selection of ci for modified files to checkin. This can mean different things for different operations, for example, it can mean all the files recursively in a directory, all the module members in a module or module hierarchy, or all the files that match a specified selector. This can be a labor-intensive option, but it can also pick up any changes that might have been forgotten.

The -iflock option only checks in files that meet certain conditions, for example, module structural changes required to preserve the integrity of the module, and locked objects. This provides a quicker checkin operation as well as security to prevent accidental modifications to unintended files.

-[no]iflock (File-based)

-[no]iflock Specifies whether to check in all modified objects in the checkin selection or only the ones that locked objects.

-The noiflock option (Default) searches the entire selection of ci for modified files to checkin. This means different things for different operations, for example, it can mean all the files recursively in a directory, or all the files that match a specified selector. This can be a labor-intensive option, but it can also pick up any changes that might have been forgotten.

The -iflock option only checks in files that are locked in the workspace. This provides a quicker checkin operation as well as security to prevent accidental modifications to unintended files.

-keep

-keep Leave a local copy of the object in the work area after checking it in. This option is the default object-state option unless a default object state has been defined (see the "fetch preference" help topic for more information).

You cannot use this option when you have enabled Link-In of large files.

Note: 'ci -keep' is equivalent to following the check-in operation with 'co -get'.

You can change whether the local object is read-only or read/write by default by using the "Check out read only when not locking" option from the Tools->Options->General dialog box in the DesignSync graphical interface, or your project leader can set this option site-wide using SyncAdmin.

-keys

-keys <mode>

Controls processing of RCS-style revision-control keywords in objects that remain in your work area after checkin. Note that keyword expansion is not the same as keyword update. For example, the $Date$ keyword is updated only during checkin; its value is not updated during checkout or populate.

Available modes are:

kkv - (keep keywords and values) The local object contains both revision control keywords and their expanded values; for example, $Revision: 1.4 $.

kk - (keep keywords) The local object contains revision control keywords, but no values; for example, $Revision$. This option is useful if you want to ignore differences in keyword expansion, such as when comparing two different versions of an object.

kv - (keep values) The local object contains expanded keyword values, but not the keywords themselves; for example, 1.4. This option is not recommended if you plan to check in your local objects. If you edit and then check in the objects, future keyword substitution is impossible, because the value without the keyword is interpreted as regular text.

ko - (keep output) The local object contains the same keywords and values as were present at check in.

Note:
- The -keys option works only with the -keep and -lock options. If you use the -share or -mirror option, keywords are automatically expanded in cached or mirrored objects, as if

the '-keys kkv' option was used.

- The EnableKeywordExpansion setting controls the expansion of keys during a check-in operation. This setting overrides the -keys option; if disabled, there is no expansion of keys, regardless of the use of the -keys option. By default, this setting is enabled; the check-in operation expands keywords. To change the default setting, your Synchronicity administrator can use the SyncAdmin tool. For information, see SyncAdmin help.

- The check-in operation detects binary files and collections and does not expand keywords when operating on these objects, even if the Enable Keyword Expansion setting is on.

-lock

-lock	Keep a locked local copy of the object in the work area after checking it in. Use this option if you want to create a new version of the object while continuing to make changes. Unless you use this option, the branch is unlocked after the check-in operation. Note: To enforce the -lock option, you have to modify the access control file for DesignSync revision control operations. See Access Control guide for more information. For examples, see the "Using access filter to Check an Action" section in the "Sample Access Controls" topic.

-mirror (File-based)

-mirror	Create a symbolic link from the work area to to the object in the mirror directory. This option requires that you have associated a mirror directory with your work area (see the "setmirror" command). Note: o This option is not supported on Windows platforms. o If you have checked in objects using the -mirror option, incremental populates by team members do not necessarily fetch the new objects until after the mirror has been updated. o The ci command does not allow this option if

```
   you use it when checking in module objects;
   in this case, ci issues an error message, but
   continues to check in other DesignSync objects.
 o When operating on a mirror directory, the
   check-in operation does not require an exact
   match between the workspace selector and the
   mirror selector in the case of <BranchName>:
   or Trunk selectors.
   The check-in operation considers
   - A selector of 'Trunk' to be the same as
     'Trunk:' and 'Trunk:Latest'
   - A selector of <BranchName>: to be the same
     as <BranchName>:Latest
```

-modulecontext (Module-based)

-modulecontext <context>	Identifies the module instance from which the checkin should occur. If no module context is provided for new files, smart module detection will attempt to identify the target module. If smart module detection cannot identify the target, use the --modulecontext to identify the target module. Note: The combination of the -modulecontext option and the -new option is mutually exclusive with the -recursive option. If you want to perform a recursive checkin with new objects in the module workspace, you must add the new objects with add and perform the ci command without the -new option. When this combination is specified, the -recursive option is silently ignored. You can also use the -modulecontext option when you are specifying a folder as the argument to be checked in. In this case, the check-in operation filters the folder, checking in only those objects that belong to the module specified with the -modulecontext option. Use -modulecontext in a recursive check-in to check in members of the specified module throughout a hierarchy. Specify an existing workspace module using the module name (for example, Chip) or a module instance name (for example, Chip%0). Note that you cannot use a -modulecontext option to operate on objects from more than one module; the -modulecontext option takes only one argument, and you can use the -modulecontext option only once on a command line.

-[no]new (Module-based)

-[no]new

Performs the initial checkin of unmanaged objects; objects that are not under revision control. By default (without -new), the check-in operation processes only managed objects or objects that have previously been added to a module with the add command. It is not an error to specify this option with managed objects or previously added module members.

The -new option is required to check in new objects you have not yet added to a module using the add command. The -new option is not needed if you have already added the objects using add. When you use the -new option to add new objects to a module, smart module detection identifies the target module, or, you can use the -modulecontext option to explicitly specify into which module the objects are to be added.

Tip: Use the -new option only if you are checking in previously unmanaged objects. Using the -new option slows the check-in process because ci must process all objects and not just the locked or modified objects.

Checking in a new object creates a new version (1.1) on a new branch (branch 1). The new version is created on the branch specified using the -branch option. If the -branch option is not specified, the version is created on the branch defined by the first selector in the persistent selector list. If the selector is not a valid branch selector (specified using the <branch>:<version> syntax), the default branch tag 'Trunk' is applied -- DesignSync expects every branch to have a tag. For example, if you apply 'setselector VaultDate(yesterday)' to a folder and then check in a previously unmanaged object from that folder, the object's new branch is tagged 'Trunk' because 'VaultDate(yesterday)' is not a valid branch tag name. See the "selectors" help topic for more details about how DesignSync resolves selectors.

Note: For a module object checkin, you cannot specify the -new option with the -recursive or the -branch options. This eliminates any issues with determining what module, module branch, or sub-module the new objects belong to.

-[no]new (File-based)

```
-[no]new          Performs the initial checkin of unmanaged objects;
                  objects that are not under revision control.
                  By default (without -new), the check-in operation
                  processes only managed objects. It is not an
                 error to specify this option with managed objects.

                  Tip: Use the -new option only if you are checking
                  in previously unmanaged objects. Using the -new
                  option slows the check-in process because ci must
                  process all objects and not just the locked or
                  modified objects.

                  Checking in a new object creates a new version
                  (1.1) on a new branch (branch 1). The new version
                  is created on the branch specified using the
                  -branch option. If the -branch option is not
                  specified, the version is created on the branch
                  defined by the first selector in the persistent
                  selector list. If the selector is not a valid
                  branch selector (specified using the
                  <branch>:<version> syntax), the default branch
                  tag 'Trunk' is applied -- DesignSync expects
                  every branch to have a tag. For example, if you
                  apply 'setselector VaultDate(yesterday)' to a
                  folder and then check in a previously unmanaged
                  object from that folder, the object's new branch
                  is tagged 'Trunk' because 'VaultDate(yesterday)'
                  is not a valid branch tag name.  See the
                  "selectors" help topic for more details about how
                  DesignSync resolves selectors.

                  The -new option also has the effect of unretiring
                  a branch when you check in a new version onto a
                  retired branch. When you unretire a branch, the
                  retire information is removed. Viewing the
                  history of the object will not show that the
                  object was retired.
```

-[no]recursive (Module-based)

```
-[no]recursive    Specifies whether to perform this operation in
                  just the specified folder or module object
                  (default) or in their subfolders/submodules.

                  If you invoke 'ci -recursive' and specify a
                  folder that is not part of a module on the
                  command line, ci operates on that folder in a
                  folder-centric fashion, checking in the modified
                  objects in the folder and its subfolders. To
                  filter the set of objects on which to operate,
                  use the -filter or -exclude options.

                  If you invoke 'ci -recursive' and specify a
```

module on the command line, ci operates on that module in a module-centric fashion, checking in all of the modified objects in the module and submodules. To filter the objects on which to operate, use the -filter or -hreffilter options. If you invoke 'ci -recursive' on a subfolder of a module and provide a module context (-modulecontext), ci recursives within the specified folder, checking in any object which is a member of the named module or one of its referenced submodules.

Note: You cannot specify the -recursive option with the -branch option when creating a new module branch.

Note: When checking in new objects to a module using the -new option, the module hierarchy is never traversed. The command checks in any unmanaged objects in a folder-centric fashion, but does not traverse any module hierarchical references, even if -recursive is specified. To determine if your work area contains new objects, use 'ls -recursive -unmanaged'. To perform the checkin recursively, use add to add the objects to the appropriate module, then run the ci command. If the referenced sub-modules are populated into the workspace, smart module detection does traverse into the folder and can correctly identify new members belonging to a submodule.

If you specify -norecursive when operating on a folder object, DesignSync does not operate on objects within that folder.

Note: The -nomodulerecursive option is no longer supported. For modules, this option is equivalent to the -norecursive option. If you specify the -nomodulerecursive option when operating on modules, ci applies the -norecursive option instead.

-recursive (Legacy-based)

-[no]recursive — Specifies whether to perform this operation in just the specified folder(s) (default) or in its subfolders also.

Note: You cannot specify the -recursive option with the -branch option when creating a new module branch.

If 'ci -recursive' is invoked on a legacy module

(or in a directory containing DesignSync REFERENCEs), ci does not traverses the directory structure or follow the REFERENCEs. The -recrusive option is not applicable to a checkin of legacy modules. If you specify a legacy module for ci, the recursive option is silently

If you specify -norecursive when operating on a folder object, DesignSync does not operate on objects within that folder.

Note: The -nomodulerecursive option is no longer supported. For modules, this option is equivalent to the -norecursive option. If you specify the -nomodulerecursive option when operating on modules, ci applies the -norecursive option instead.

-recursive (File-based)

-[no]recursive — Specifies whether to perform this operation in just the specified folder(s) (default) or in its subfolders also.

Note: You cannot specify the -recursive option with the -branch option when creating a new module branch.

If you specify -norecursive when operating on a folder object, DesignSync does not operate on objects within that folder.

-reference

-reference — Keep a reference to the object in the work area after the check-in operation. A reference does not have a corresponding file on the file system but does have DesignSync metadata that makes it visible to Synchronicity programs. References are useful when you want to have the complete context of the objects in a project, but do not need the objects locally.

Note: Synchronicity recommends against using the -reference option when operating on a collection object. If you use the -reference option, DesignSync creates a reference in the metadata for the collection object but member files are not processed and are not included in the metadata.

-report

-report error| brief|normal| verbose	Controls the amount and type of information displayed by ci command. The information each option returns is discussed in detail in the "Understanding the Output" section above. error - lists failures, warnings, and success failure count. brief - lists failures, warnings, module create/remove messages, some informational messages, and success/failure count. normal - includes all information from brief and lists all the updated objects, and messages about objects excluded by filters from the operation. (Default) verbose - provides full status for each object processed, even if the object is not updated by the operation.

-[no]resume (Module-based)

-[no]resume	Specifies whether to perform a recovery check-in. Specify the -resume option (the default) if a previous recursive check-in of a module has failed. This option causes the check-in to continue from the point where the failure occurred. Specify the -noresume option to start the check-in from scratch. Note: If a module checkin fails, and the check in operation contains structural changes, such as moved or removed module members, the subsequent checkin always starts from scratch. The -resume option is silently ignored.

-[no]retain

-[no]retain	Indicates whether to retain the 'last modified' timestamp of the objects that remain in your work area. If you are using the -share option or a mirror is set on the workspace, then this also applies to the object put into the file cache or mirror. The links for the cache or mirror in your work area use the link creation time as the "last modified" time. If you specify the -reference option, no object

is created in your work area, so there is no timestamp information at all.

If you do not specify '-retain' or -noretain', the ci command follows the DesignSync registry setting for Retain last-modification timestamps. By default, this setting is not enabled; therefore, the timestamp of the local object is the time of the check-in operation. To change the default setting, your Synchronicity administrator can use the SyncAdmin tool. For information, see SyncAdmin Help.

The mirror system by default fetches objects into the mirror with the -retain option. The mirror administrator can configure mirrors to use the -noretain option. The default setting should agree with the Retain last-modification timestamp registry setting to maintain consistency. See the "Mirror Administration Server Registry Settings" topic for setting of the co or populate options for mirrors.

-[no]retry (Module-based)

-[no]retry

Specifies whether, if the module checkin fails, DesignSync attempts a retry of the checkin.

-retry attempts to retry the checkin if the retryOnModuleCiFailureHook is enabled and the module meets the conditions defined within the hook for retry; or the checkin failure was due to a communication connect failure and the ModuleFailureRetryAttempts registry setting is set to a non-zero value, indicating one or more retries. The checkin will be retried as long as a communication connect failure is still the cause of failure and the number of checkin retries for this module has not surpassed the ModuleFailureRetryAttempts value. (Default)

For more information on the retryOnModuleCiFailureHook or the ModuleFailureRetryAttempts registry keys, see the Administrator's Guide.

-noretry does not attempt to retry to checkin. If the module checkin fails, the operation fails for that module. If it is part of a hierarchical module checkin, the checkin continues attempting to checkin the next module.

-[no]selected

-[no]selected	Specifies whether to perform this operation on objects in the select list (see the "select" command), as well as the objects specified on the command line. If no objects are specified on the command line, -selected is implied. By default (-noselected), ci operates only on the objects specified on the command line.

-share

-share	Put a copy of the object in the file cache directory, and create a link in the work area to that cached object. Note: This option is not supported on Windows platforms. If you use 'ci -share' on a collection object, DesignSync checks in the symbolic link, unless it is a link to a cache.

-[no]skip (Module-based)

-[no]skip	Specifies whether to check in the version even if it is not derived from the Latest version (the branch contains higher-numbered versions). By default (-noskip), versions are not skipped. This situation can occur when you check out the Latest version without a lock and other users check in new versions prior to your checkin, or when you have intentionally checked out an older version of the object. You can use the -skip option with module members, to skip previous checkins of module members. The -skip option does not skip module versions. Any structural module changes made in the versions between the version populated in the workspace and the version created appear in the new version. You also typically need -skip when using -branch to check into a branch other than the current branch. See -branch for details. You must specify -force if the version you are checking in is not locally modified. You must have local copies of the file versions that you want to check in. You cannot have links

to a cache.

If the server contains structural changes, for example removed or moved files, that are not reflected in the workspace, you will be unable to perform a checkin, even with the -skip option.

Cautions:

- Changes in skipped versions are not reflected in the new Latest version and are effectively lost. Use the -skip option when you are intentionally promoting an older version of an object to be the Latest (similar to a rollback operation) If you are using modules, there is a modules rollback command that allows you to create a new module version from an older version, skipping all structural and module member changes..

- Use caution when you use -skip with module objects because the -skip option does not override intervening changes to the structure of a module. This means you may be unaware of structural module changes that have occurred that do not conflict with your actions, for example new, modified, or removed objects.

-[no]skip (File-based)

-[no]skip

Specifies whether to check in the version even if it is not derived from the Latest version (the branch contains higher-numbered versions). By default (-noskip), versions are not skipped.

This situation can occur when you check out the Latest version without a lock and other users check in new versions prior to your checkin, or when you have intentionally checked out an older version of the object.

You also typically need -skip when using -branch to check into a branch other than the current branch. See -branch for details.

You must specify -force if the version you are checking in is not locally modified.

You must have local copies of the file versions that you want to check in. You cannot have links to a cache or mirror directory.

Caution: Changes in skipped versions are not

reflected in the new Latest version and are effectively lost. Use the -skip option when you are intentionally promoting an older version of an object to be the Latest.

-tag (Module-based)

-tag <tagname>

Tags the object version or module version on the server with the specified tagname.

For module objects, all objects are evaluated before the checkin begins. If the objects 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 checkin fails.

For more information on access controls, see the ENOVIA Synchronicity Access Control Guide. For more information on version tags, see the tag command.

Note: The -tag option will not work on modules stored on DesignSync server versions prior to V6R2008-1.0.

-tag (File-based)

-tag <tagname>

Tags the object version on the server with the specified tagname.

If the user does not have access to add a tag, the object is checked in without a tag.

For more information on access controls, see the ENOVIA Synchronicity Access Control Guide. For more information on version tags, see the tag command.

-trigarg

-trigarg <arg>

Specifies an argument to be passed from the command line to the triggers set on the check-in operation. If the argument contains whitespace, enclose the argument within double quotation marks ("") if using the dss command shell or braces ({}) if using the stcl command shell.

--

```
--                  Indicates that the command should stop looking
                    for command options. Use this option when an
                    argument to the command begins with a hyphen (-).
```

RETURN VALUE

```
In dss/dssc mode, you cannot operate on return values, so the return
value is irrelevant.

In stcl/stclc mode, two lists are returned, where the first list
is non-empty if at least one object was successfully processed,
and the second list is non-empty if at least one object failed.

Notes:
 - "successfully processed" may not mean "successfully checked
   in". For example, attempting to check in an object that is not
   locally modified is considered a success even though no new
   version is created.
 - Scripts should only test for non-empty lists to
   determine success or failure. The actual content of a non-empty
   list currently takes the form "Objects succeeded (n)" and
   "Objects failed (n)", where 'n' is the number of objects that
   succeeded or failed. However, these return values are subject to
   change in a future release.
 - If all objects fail, an exception occurs (the return value is
   thrown, not returned).
 - Objects reported as "excluded by filter," may have been excluded
   either with the -filter option (for modules) or with the -exclude
   option.
 - If a comment editor is defined, but cannot be used, the command
   automatically switches to the interactive command-line comment
   mode.
```

SEE ALSO

```
caching, cancel, co, populate, select, setmirror, setselector,
selectors, swap, tag, command defaults, unlock
```

EXAMPLES

- Example of Creating a Module and Performing an Initial File Checkin (Module-based)
- Example of Checking in Module Structure Changes (Module-based)
- Example of Checking in on a New Branch (Module-based)
- Example of Attempting to Modify A Member in a Static Workspace (Module-based)
- Example of Checking in a File without a Comment (File-based)
- Example of Checking in New Files (File-based)
- Example of Checking in Recursively (File-based)

- Example of a Dry-Run Checkin Showcasing Wildcard Usage (File-based)
- Example of Checkin to a Branch (File-based)

Example of Creating a Module and Performing an Initial File Checkin (Module-based)

```
The following example creates a module, Chip, version 1.1, adds
 module members, chip/makefile, chip/verilog/chip.v, and
 DOC/Chip.doc, and finally checks the members in, thus generating
 a new module version, 1.2:
   stcl> mkmod -comment "The main chip" \
      sync://mysrvr:2647/Modules/Chip \
      -path /home/karen/MyModules
   stcl> add -recursive Chip chip DOC
   stcl> ci -keep -nocomment Chip
```

Example of Checking in Module Structure Changes (Module-based)

```
 The following example shows a checkin on a workspace that has
 renamed, removed, and added

 stcl> ci -nocomment Chip%1

  Beginning Check in operation...


  Checking in objects in module Chip%1

  Total data to transfer: 0 Kbytes (estimate), 10 file(s), 0 collection(s)
  Progress: 0 Kbytes, 0 file(s), 0 collection(s), 0.0% complete
  Progress: 3 Kbytes, 10 file(s), 0 collection(s), 100.0% complete
  Progress: 3 Kbytes, 10 file(s), 0 collection(s), 100.0% complete

  Checking in: /chip.bat                      Success - Renamed from
/chip.exe
  Checking in: /doc/chip.doc                  Success - New version: 1.1
  Checking in: /doc/commands.html             Success - Removed

  Chip%1: Version of module in workspace updated to 1.8

  Finished checkin of Module Chip%1, Created Version 1.8

  Time spent: 1.2 seconds, transferred 3 Kbytes, average data rate 2.5 Kb/sec

  Checkin operation finished.

   {Objects succeeded (3)} {}
```

Example of Checking in on a New Branch (Module-based)

```
 The following example creates a new branch from a module. In this
 example, one file was modified for the new version and another was
```

```
added.

Note: The workspace selector changes to the new branch when you run
the checkin.

stcl> ci -nodefaults -keep -retain -hrefversions -exclude *.log
   -branch Beta -keys kkv -nocomment -report normal Chip%1

 Beginning Check in operation...

 Chip%1: Creating branch Beta

 Checking in objects in module Chip%1

 Total data to transfer: 10 Kbytes (estimate), 2 file(s), 0
 collection(s)
 Progress: 0 Kbytes, 0 file(s), 0 collection(s), 0.0% complete
 Progress: 4 Kbytes, 1 file(s), 0 collection(s), 45.7% complete
 Progress: 11 Kbytes, 2 file(s), 0 collection(s), 100.0% complete

 Checking in: Chip%1\chip.docx              Success - New version:
 1.1.1.1
 Checking in: Chip%1\chipsub.c              Success - New version:
 1.1

 Chip%1: Version of module in workspace updated to 1.11.1.2
 Chip%1: Selector of module in workspace updated to Beta:


 Finished checkin of Module Chip%1, Created Version 1.11.1.2

 Time spent: 0.4 seconds, transferred 11 Kbytes, average data rate
 26.1 Kb/sec
 Checking in: \chip.c                       : Success - No new
 version created. Lock Removed.
 Checking in: \chip.h                       : Success - No new
 version created. Lock Removed.

 Checkin operation finished.

 {Objects succeeded (4)} {}
```

Example of Attempting to Modify A Member in a Static Workspace (Module-based)

```
DesignSync does not allow modifications in workspaces that have been
populated in static mode. The following example shows what happens
when you modify and attempt to check in a workspace that has been
populated with a static module selector.

dss> populate -rec -version Gold Chip-R419%0
 Beginning populate operation at Fri Oct 28 12:41:08 Eastern Daylight
 Time 2016...

 Setting Selector [Gold] on workspace module c:\workspaces\ChipDev419
```

```
   \chip\Chip-R419%0
  WARNING: Chip-R419%0: Changing the selector to a static value (Gold).
  You will not be able to check in module or member modifications.

  Selector on module c:\workspaces\ChipDev419\chip\Chip-R419%0 was
  modified.
...
  Finished populate operation.
  ##### WARNINGS and FAILURES LISTING #####
  #
  # WARNING: Chip-R419%0: Changing the selector to a static value
  # (Gold).
  # You will not be able to check in module or member modifications.
  ########################################
  {Objects succeeded (6)} {Objects failed (2)}

dss> ci -comment "Checking in changes" Chip-R419%0

  Beginning Check in operation...
  Chip-R419%0: Cannot checkin module with static selector.

  Checkin operation finished.
  {} {}
```

Note: If you have data that you need to check in, you should either change the selector for the workspace to dynamic selector, or move the modified data to a workspace that has the module populated with a dynamic selector.

Example of Checking in a File without a Comment (File-based)

The following example checks in the file 'pcimaster.vbh' without a comment, leaving a link to the cache in the work area:

```
  stcl> ci -nocomment -share pcimaster.vbh
```

Example of Checking in New Files (File-based)

The following example uses the -new option to check in previously unmanaged files 'alu.v' and 'decoder.v'. A check-in comment is provided. Note that no state option is specified, so the default fetch preference is used (or -keep, the default for ci, if no default fetch preference is defined):

```
  stcl> ci -comment "Beta versions" -new alu.v decoder.v
```

Example of Checking in Recursively (File-based)

The following example recursively checks in an entire work area, while retaining locks on the objects:

```
  stcl> ci -rec -nocomment -lock .
```

Example of a Dry-Run Checkin Showcasing Wildcard Usage (File-based)

```
The following example performs a dryrun checkin because of the
complex wildcarding used to specify the objects to be checked in:
  stcl> ci -dryrun -recursive -exclude *.vg,*.log pci*.* alu
```

Example of Checkin to a Branch (File-based)

```
This example shows the typical "project branching" approach to
working with multiple branches. The project leader, who has a work
area containing all objects in the project, creates new branches
off the current versions, sets the persistent selector list to use
the new branch, then checks into the new branch. Team members set
their selector lists to point to the new branch and then populate.
Project Leader:
  stcl> mkbranch -rec Rel2.1 .
  stcl> setselector -rec Rel2.1:Latest .
  stcl> ci -com "Creating 1st version on Rel2.1 branch" top.v
Team Members:
  stcl> setselector -rec Rel2.1:Latest .
  stcl> populate
```

CO

co Command

NAME

```
co                  - Checks out the specified objects
```

DESCRIPTION


```
This command checks out the specified objects (files or
collections). A checkout retrieves a working copy of a specified
version of an object from the revision-control vault and places it
in your work area.

Important: You must use the populate command rather than the
co command when fetching modules or module objects. The co
```

```
command does not support modules.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

This command supports the command defaults system.
```

Object States

```
Upon checking out the specified objects, DesignSync changes the
state of all processed objects--the objects that need an update,
as well as the up-to-date objects, as follows:
  1. DesignSync obeys the state option (-get, -lock, -reference,
     -share, -mirror) specified on the command line.
       Note: If objects are designed uncachable, the -share and
       -mirror states are silently ignored and the objects are
        populated with -keep. For more information, see the caching
        commands.
  2. If no state option is specified, DesignSync uses the default
     fetch state as specified by your project leader. See
     the "fetch preference" help topic for more information.
  3. If a default fetch state is not defined, the default
     behavior for 'co' is -get.

DesignSync changes the states of all objects being processed,
even if they are already up-to-date.  Specify the -nounifystate
option if you want to change the state of only the objects
that need an update.

If your team is using the locking design methodology:
  o You check out an object with a lock (-lock option) if you plan to
    edit the object. DesignSync locks the branch associated with
    the version you are checking out, prohibiting other team
    members from creating new versions on that branch. You,
    the holder of the lock, reserve the right to create the
    next version on that branch.
  o You "fetch" -- check out without a lock -- an object if you want
    read-only access to that object. You use the -get, -mirror, or
    -share option depending on your team's sharing methodology and
    the platform you are on (-mirror and -share are only available
    on Unix platforms).

If your team is using the nonlocking, or merging, design
methodology, you always fetch objects, even if you plan to edit
them. If you and another team member edit copies of the same
object, the first person to check in the object creates the next
version. The other person must first merge (-merge option) the
changes from this new Latest version into his or her local copy,
manually resolve any merge conflicts, then check in the merged
object.
```

Determining the Objects to be Checked Out

```
Arguments to the 'co' command must be versionable objects (files
and collections), or local folders (only meaningful when you use
the -recursive option).

DesignSync determines what version of a design object you want to
check out as follows:
  1. DesignSync obeys the selector list specified by the -version
     option.
  2. If -version is not specified, DesignSync uses the
     object's persistent selector list. The default persistent
     selector is 'Trunk', in which case DesignSync checks out
     the Latest version from Trunk.

See the "selectors" help topic for details on selectors,
selector lists, and persistent selector lists.

In multi-branch environments, you use the 'co' and 'populate'
commands to merge branches. In most cases, each new branch that is
created is eventually merged back into the main development
branch. See the -merge and -overlay options for more information
on merging.

Note: If you specify a collection member as the object to be
operated on, DesignSync skips the object and warns that the
object is not versionable. If DesignSync attempts to operate on
a collection member specified implicitly (through the use of
wildcards or a recursive operation), DesignSync silently skips
the object. You can change this behavior by using the SyncAdmin
"Map operations on collection members to owner" setting. If you
select this setting and DesignSync attempts to operate on a
collection member during a revision control operation,
DesignSync determines the member's owner collection and
operates on the collection as a whole.
```

Checking Out Objects with Different Version Selectors

```
When checking out multiple objects with different version selectors
in a single operation, you can specify the version number or the
branch selector of the objects you are checking out as
an object;selector specification.  The selector specified (in
an object;selector) can be any legal selector, such as:
in a single operation, specify the version number or the branch
 o Trunk:Latest (branch selector)
 o golden (version selector)
 o auto
 o selector list (comma separated)
   Note: The selector specified in an object;selector will always
         override any selector previously set (via setselector).
         The object (in an object;selector) can be:
 o A relative path (in which case 'pwd' will get prefixed)
 o A URL
 o An absolute path
to the co command. Instead of only accepting an object to operate on,
```

Note: You can perform "co object;selector" only on files and not folders.

Checkout Versus Populate

The 'co' and 'populate' commands are similar in that they retrieve versions of objects from their vaults and place them in your work area. They differ in several ways, most notably:

- The 'co' command operates on objects that you already have locally, whereas 'populate' updates your work area to reflect the status of the vault. For example, you use 'co' to change the object state or to get a different version of a local object. You use 'populate' to retrieve versions of newly managed objects created by other team members. Note that for 'co', the object does not actually need to be local if you specify the exact name of the object (no wildcards are allowed). For example, 'co top.v' succeeds if the vault contains 'top.v'. However, it is more typical to use 'populate' to retrieve objects that are not currently in your work area.
- You must use the populate command rather than the co command when fetching modules or module objects. The co command does not support modules.
- The 'co' command considers the persistent selector list for each object that is checked out, whereas 'populate' only considers the persistent selector list for the top-level folder that is being populated.

How the Check-Out Operation Handles Collections with Local Versions

For collection objects that have local versions (for example, custom generic collections), the check-out operation handles local versions in the following way.

When you check out a collection object, the check-out operation removes from your workspace any local version of the object that is unmodified. (Because these local versions exist in the vault, you can refetch them.) The operation then fetches from the vault the specified collection object (with the local version number it had at the time of checkin).

If the current local version in your workspace is modified, the check-out operation fails unless you specify 'co -force'. (The -force option lets the local version with the modified data be replaced with the local version of the object you are checking out.) Note: The current local version is the one with the highest local version number. DesignSync considers a local version to be modified if it contains modified members or if it is not the local version originally fetched from the vault when the collection object was checked out or populated to your workspace.

```
The -savelocal option tells the check-out operation what to do with
local versions in your workspace other than a current local version
that is modified. For information, see OPTIONS.
```

Auto-Branching

```
You can create a new, locked branch by using 'co -lock' with a
selector and autobranching. This branch can be unlocked without
creating a new version by:

 o Using 'cancel' from the workspace where the branch was locked.
 o Using 'unlock' on the vault.
 o Using 'ci' from the workspace where the branch was locked, without
   making modifications.

In these cases, the lock is removed from the vault, the auto-created
branch is removed, and the branch tag is deleted. If the branch is
removed but still exists in the metadata of a workspace, some commands
(such as the 'url' commands and 'vhistory') will fail with "No such
version.
```

SYNOPSIS

```
co  [-[no]comment "text"] [-[no]force] [-exclude <object>[,<object>...]]
    [[-lock [[-keys <mode>] | [-from {local | vault}]] [-merge]
    [-reference]] | [-get  [[-keys <mode>] | [-from {local | vault}]]
    [-overlay <selector>[,<selector>...]]] | -share  |  -mirror |
     -reference] [-merge] [-[no]recursive] [-[no]retain]
    [-savelocal <value>] [-[no]selected] [-trigarg <arg>]
    [-[no]unifystate] [-version <selector>[,<selector>...]]
    [--] [<argument> [<argument...]]
```

ARGUMENTS

- DesignSync Object
- DesignSync Folder

DesignSync Object

```
<DesignSync object> Fetches the object from its vault.
```

DesignSync Folder

```
<DesignSync folder> Fetches the contents of the specified folder.
                    You can also use the -dir option to specify
                    a folder to be fetched.
```

OPTIONS

-comment

-[no]comment "text"	Specifies whether to check out the specified objects with a description attached. By default (-comment), co requires a comment. If you specify -nocomment, you can still provide comments when you check in the objects. Enclose the description in double quotes if it contains whitespace. When you check in the objects, DesignSync appends check-in comments, if there are any, to the check-out comments to create a "version log". The ampersand (&) and equal (=) characters are replaced by the underscore (_) character in revision control notes. If you specify -comment without specifying text for the comment, or if -comment is set as the default, DesignSync prompts for a comment either interactively on the command line or with the defined file editor. For more information about using the defined file editor, see the DesignSync Data Manager Administrator's Guide, "General Options."

-exclude

-exclude <objects>	Specifies a comma-separated list of objects (files, collections, folders) to be excluded from the operation. Wildcards are allowed. Do not specify paths in your arguments to -exclude. Before operating on each object (such as during a recursive checkin), DesignSync compares the object's leaf name (path stripped off) to the exclude list to see if there is a match. Because the object's path is removed, the object will not match any object in the exclude list specified with a path. For example, if you specify '-exclude bin/*.exe', you will not successfully exclude bin/foo.exe or any other *.exe file. You need to instead specify '-exclude *.exe', or '-exclude foo.exe' if you want to exclude only 'foo.exe'. The result is that you cannot exclude a specific instance of an object -- you exclude all matching objects. In addition to objects you specify using the -exclude option, the "These objects are always excluded" field from the Tools->Options-> General->Exclude Lists dialog box in DesignSync lists objects that are always excluded from revision-control operations.

-force

-[no]force	Indicates whether to allow the version being checked out to overwrite a locally modified copy of the object. The default (-noforce) prevents overwriting of locally modified objects. Use this option with caution, because changes in the local copy are lost. Using -force with -unifystate changes the state of all objects including locally modified objects, in which case, local modifications are overwritten and objects are fetched according to the specified state or the default fetch state.

-from

-from <where>	Specifies whether the object is fetched from the vault ('-from vault') or from the cache or mirror ('-from local'). By default, DesignSync fetches from the cache or

mirror ('-from local'), a performance optimization specific to the 'co -lock', 'co -get', 'populate -lock', and 'populate -get' commands. For details, see the Performance Optimization Overview in the DesignSync Data Manager Administrator's Guide. Note that this option is silently ignored when the optimization is not possible, including when the -keys option is specified.

-get

-get — Fetch an unlocked copy.

You can change whether the local object is read-only (typical when using the locking model) or read/write (typical when using the merging model) by default by using the "Check out read only when not locking" option from the Tools->Options->General dialog box in the DesignSync graphical interface. Your project leader can also set this option site-wide using SyncAdmin.

This option is the default object-state option unless a default fetch preference has been defined (see the "fetch preference" command-line topic for more information).

-keys

-keys <mode> — Controls processing of RCS-style revision-control keywords in objects during checkout. Note that keyword expansion is not the same as keyword update. For example, the $Date$ keyword is updated only during checkin; its value is not updated during checkout or populate.

Available modes are:

kkv - (keep keywords and values) The local object contains both revision control keywords and their expanded values; for example, $Revision: 1.4 $.

kk - (keep keywords) The local object contains revision control keywords, but no values; for example, $Revision$. This option is useful if you want to ignore differences in keyword expansion, such as when comparing two different

versions of an object.

kv - (keep values) The local object contains expanded keyword values, but not the keywords themselves; for example, 1.4. This option is not recommended if you plan to check in your local objects. If you edit and then check in the objects, future keyword substitution is impossible, because the value without the keyword is interpreted as regular text.

ko - (keep output) The local object contains the same keywords and values as were present at check in.

Note:
- The -keys option works only with the -get and -lock options. If you use the -share or -mirror option, keywords are automatically expanded in cached or mirrored objects, as if the '-keys kkv' option was used.

- The EnableKeywordExpansion setting controls the expansion of keys during a check-out operation. This setting overrides the -keys option; if disabled, there is no expansion of keys, regardless of the use of the -keys option. By default, this setting is enabled; the check-out operation expands keywords. To change the default setting, your Synchronicity administrator can use the SyncAdmin tool. For information, see SyncAdmin help.

- The check-out operation detects binary files and collections and does not expand keywords when operating on these objects, even if the Enable Keyword Expansion setting is on.

-lock

-lock

Lock the branch after retrieving the specified version from the vault. Only the user who has the lock can check in a newer version of the object on that branch. Use this option when your project team uses the locking model (as opposed to the merging model) and you intend to make changes to an object. Use the 'ci' or 'cancel' command to remove the lock.

You can use the -lock option to acquire a lock for an object you have already edited. In this case, DesignSync locks the object and retains your edited version without overwriting it.

DesignSync only locks the object if there have been no other changes checked in for the object and if no other users have acquired a lock on the object since you last fetched it.

Note: If you specify 'co -lock', then by default the check-out operation also uses the '-from local' option. The result is that the check-out operation locks the object in the vault and keeps local modifications in your workspace. See the -from option for information.

Use the -lock option with the -reference option to check out a locked reference. Locked references are useful if you intend to generate an object and want to lock it before regenerating, as opposed to editing the previous version. Upon generation of the object, it automatically becomes a locked copy rather than a locked reference. Getting locked references for generated objects is faster because DesignSync does not fetch the previously generated object. If the object exists already in the workspace, DesignSync deletes it. If the object exists and is locally modified, the operation fails. If you intend to overwrite the modifications, use -force to create the locked reference. If the default fetch state is 'reference' and you specify the -lock option without the -reference option, DesignSync leaves locked copies of the objects in your workspace; you must explicitly apply the -reference option with the -lock option if you want locked references in your workspace.

-merge

-merge

Fetch the Latest version of the object on the branch specified by the persistent selector list, and merge it with the current, locally modified version. If you are not doing an overlay merge (see -overlay) and the current version is not locally modified, the -merge defaults to a -get and fetches the new version without merging. By definition, a merge expects a locally modified object, so the -force option is not required.

Merging is not supported for collection objects (for example, Cadence collections).

The -merge option supports the merging work model (as opposed to the locking work model)

```
where multiple team members can check out and
edit the Latest version concurrently. The
first team member to check in creates the next
version. Other team members must merge the new
Latest version into their local copy before
they can check in their changes.

If there are no merge conflicts, then the merge
succeeds, leaving you the merged file in
your work area. If there are conflicts,
DesignSync issues a warning, and you must edit
the merged file to resolve the conflicts
before DesignSync allows you to check in the
merged version. Conflicts are shown as
follows:
  <<<<<<< local
  Lines from locally modified version
  =======
  Lines from selected server version
  >>>>>>> versionID

DesignSync considers the conflicts resolved
when the file no longer contains any of the
conflict delimiters (exactly 7 less-than,
greater-than, or equal signs starting in
column 1). The status of an object, as
displayed by 'ls' or from the List View in the
DesignSync graphical interface, indicates if
conflicts exist. The 'url inconflict'
command also determines whether a file has
conflicts.

Most merges are between two versions on the
same branch; the current branch and
the branch specified by the persistent
selector list are typically the same. However,
a merge can also be performed across branches
by setting the persistent selector list to a
different branch. Following the merge, you are
on the branch associated with the version
specified by the persistent selector
list. This is a "merge to" operation. If you
want to stay on the current branch instead,
use the -overlay option. Overlay, or "merge
from", merges are more common when merging
branches. See the -overlay option for details.

Notes:
o Although -merge is intended to be used
  with the merging work model, you can
  specify -lock with -merge to lock the
  branch as part of the merge operation.
o The -merge option implies -get unless you
  specify -lock. You can also explicitly
  specify -get.
o The -merge option is mutually exclusive with
  -mirror and -share.
```

```
                    o The -version and -merge options are mutually
                      exclusive unless you specify
                      '-version Latest'.
```

-mirror

```
 -mirror            Create a symbolic link from the work area
                    to the object in the mirror directory. This
                    option requires that you have associated a mirror
                    directory with your work area (see the setmirror
                    command). In addition, the effective workspace
                    selector (set using 'setselector', 'setvault',
                    or the -branch option) must match the mirror
                    workspace selector.

                    Note:
                    o This option is not supported on
                      Windows platforms.
                    o When operating on a mirror directory, the
                      check-out operation does not require an exact
                      match between the workspace selector and the
                      mirror selector in the case of <BranchName>:
                      or Trunk selectors.
                      The check-out operation considers
                      - A selector of 'Trunk' to be the same as
                        'Trunk:' and 'Trunk:Latest'
                      - A selector of <BranchName>: to be the same
                        as <BranchName>:Latest
```

-overlay

```
 -overlay <selectors> Replace your local copies with the versions
                      specified by the selector list. The
                      current-version status, as stored in local
                      metadata, is unchanged. For example, if you
                      have version 1.5 (the Latest version) of a
                      file and you overlay version 1.3, your current
                      version is still 1.5. You could then check in
                      this overlaid version. This rollback
                      operation is equivalent to checking out
                      version 1.3, then using 'ci -skip' to check in
                      that version.

                      Note: The overlay operation overlays specified
                      local copies even if you checked out those
                      versions with a lock.

                      Typically, you use -overlay with -merge to
                      merge the two versions instead of overlaying
                      one version on another. The combination of
                      -overlay and -merge lets you merge from one
                      branch to another, the recommended method for
```

merging across branches. Following the overlay merge, you are working on the same branch as before the operation.

The -overlay and -version options are mutually exclusive. You specify the version you want to to overlay as an argument to the -overlay option. The -version option always updates the 'current version' information in your work area, which is not correct for an overlay operation.

Note:
- To use -overlay to specify a branch, specify both the branch and version as follows: '<branchtag>:<versiontag>', for example, 'Rel2:Latest'. You can also use the shortcut, '<branchtag>:', for example "Rel2:". If you don't explicitly specify the branch selector in this way, DesignSync does not resolve the selector as a branch selector.
- The -overlay option implies -get, but you can also explicitly specify -get.
- The -overlay option is mutually exclusive with -mirror, -share, -lock, and -version options.

-[no]recursive

-[no]recursive

Indicates whether to perform this operation on all objects in all subfolders of the specified folders. This behavior (-recursive) is the default behavior of the graphical interface, but not of the command line interface.

-reference

-reference

Create a reference to an object in the vault. A reference does not have a corresponding file on the file system but does have local metadata that makes the reference visible to Synchronicity programs. Create a reference when you want your work area to reflect the contents of the vault but you do not need a physical copy. Use the -reference option with the -lock option to create a locked reference. Locked references are useful if you intend to generate an object and want to lock it before regenerating, as opposed to editing the previous version.

Note: Synchronicity recommends against using the -reference option when operating on a collection object. If you use the -reference

option, DesignSync creates a reference in the metadata for the collection object but member files are not processed and are not included in the metadata.

-[no]retain

-[no]retain

Indicates whether to retain the 'last modified' timestamp of the checked-out objects as recorded when the object was checked into the vault. If the workspace is set to use a mirror, or the checkout is run using -share, this will also apply to the object placed in the mirror or LAN cache if the object doesn't already exist in the mirror or cache. The links in your work area to the cache or mirror have timestamps of when the links were created.

If you specify the -reference option, no object is created in your work area, so there is no timestamp information at all.

If an object is checked into the vault and the setting of the -retain option is the only difference between the version in the vault and your local copy, DesignSync does not include the object in checkout operations.

If you do not specify '-retain' or -noretain', the co command follows the DesignSync registry setting for Retain last-modification timestamps. By default, this setting is not enabled; therefore, the timestamp of the local object is the time of the check-out operation. To change the default setting, your Synchronicity administrator can use the SyncAdmin tool. For information, see SyncAdmin Help.

The mirror system, by default,fetches objects into the mirror with the -retain option. The mirror administrator, however, can define mirrors to use the -noretain option. The default setting should agree with the Retain last-modification timestamp registry setting to maintain consistency. See the "Mirror Administration Server Registry Settings" topic for setting of the co or populate options for mirrors.

Note: When fetching from the cache or mirror (by specifying the '-from local' option), the last modified timestamp comes from the file in the cache or mirror, not from the version that was

checked into the vault. If the file was fetched into the cache or mirror with the -retain option, these two timestamps are the same. But if the file was fetched into the cache or mirror with the -noretain option and then fetched into the workspace with both the '-from local' and '-retain' options, the 'last modified' timestamp used is the time the object was fetched into the cache or mirror.

-savelocal

-savelocal <value> This option affects collections that have local versions.

When it checks out an object, the check-out operation first removes from your workspace any local version that is unmodified. (To remove a local version containing modified data, specify 'co -force'.) Then the check-out operation fetches the object you are checking out (with the local version number it had at the time of checkin).

The -savelocal option specifies the action that the check-out operation takes with modified local versions in your workspace (other than the current, or highest numbered, local version). (DesignSync considers a local version to be modified if it contains modified members or if it is not the local version originally fetched from the vault when the collection object was checked out or populated to your workspace.)

Specify the -savelocal option with one of the following values:

save - If your workspace contains a local version other than the local version being fetched, the check-out operation saves the local version for later retrieval. See the 'localversion restore' command for information on retrieving local versions that were saved.

fail - If your workspace contains an object with a local version number equal to or higher than the local version being fetched, the check-out operation fails. This is the default action.

Note: If your workspace contains an object with local version numbers lower than the local version being fetched and if these local

versions are not in the DesignSync vault, the check-out operation saves them. This behavior occurs even when you specify '-savelocal fail'.

delete - If your workspace contains a local version other than the local version being fetched, the check-out operation deletes the local version from your workspace.

If you do not specify the -savelocal option, the check-in operation follows the DesignSync registry setting for SaveLocal. By default, this setting is "Fail if local versions exist" ('-savelocal fail'). To change the default setting, a Synchronicity administrator can use the Command Defaults options pane of the SyncAdmin tool. For information, see SyncAdmin Help.

Note:
- You may need to use the -force option with the -savelocal option to allow the object being checked out to overwrite a locally modified copy of the object. For an example scenario, see EXAMPLES.
- The -savelocal option only affects objects of a collection defined by the Custom Type Package (CTP). This option does not affect objects that are not part of a collection or collections that do not have local versions.

-[no]selected

-[no]selected — Indicates whether to perform this operation on objects in the select list (see the "select" command), as well as the objects specified on the command line. By default, (-noselected), co does not use the select list. If no objects are specified on the command line, the -selected option is implied.

-share

-share — Fetch a shared copy. Shared objects are stored in the file cache directory and a link to the cached object is created in the work area.

Note: This option is not supported on Windows platforms.

-trigarg

```
 -trigarg <arg>       Specifies an argument to be passed from the
                      command line to the triggers set on the
                      check-out operation. If the argument contains
                      whitespace, enclose the argument within
                      double quotation marks ("") if using the dss
                      command shell or braces ({}) if using the stcl
                      command shell.
```

-[no]unifystate

```
 -[no]unifystate      Sets the state of all objects processed,
                      even up-to-date objects, to the specified state
                      (-get, -lock, -share, -mirror, or -reference) or
                      to the default fetch state, if no state option is
                      specified.  See the "fetch preference" help topic
                      for more information.

                      By default, 'co' changes the state of all
                      processed objects that need an update
                      as well as up-to-date objects (-unifystate). If
                      the -nounifystate option is specified, DesignSync
                      changes the state of only the objects that need
                      an update.

                      The -unifystate option does not change the state
                      of locally modified objects; use -force with
                      -unifystate to force a state change, thus
                      overwriting local modifications. The -unifystate
                      option does not cancel locks; you can check in
                      the locked files, use the 'cancel' command to
                      cancel locks you have acquired, or use the
                      'unlock' command to cancel team members' locks.

                      Note: The -unifystate option is ignored when
                      you lock design objects.  If you check out
                      locked copies or locked references, DesignSync
                      leaves all processed objects in the requested
                      state.
```

-version

```
 -version <selector>  Specifies the versions of the objects to be
                      checked out. The selector list you specify
                      (typically a version or branch tag) overrides
                      the object's persistent selector list, but the
                      persistent selector list is not modified.

                      If you specify a date selector (Latest or
                      Date(<date>)), you must specify a branch or
                      version with it. For example, you want to check
                      out the latest version of 'Gold:,Trunk' specify
                      'co -version Latest'.
```

If you specify the specific version number for the Latest version of objects on a retired branch, the co command fetches the objects into your workspace. If you specify the version as a branch selector for a retired branch, the co command does not fetch the specified retired files.

Note:
- If the selector you specify using -version does not exactly match the work area selector, your next populate will be automatically forced to be full (non-incremental). See the 'populate' command description for more information.
- To use -version to specify a branch, specify both the branch and version as follows: '<branchtag>:<versiontag>', for example, 'Rel2:Latest'. You can also use the shortcut, '<branchtag>:', for example "Rel2:". If you don't explicitly specify the branch selector in this way, DesignSync does not resolve the selector as a branch selector.
- Specify '-version Latest' only if necessary because in some cases, DesignSync augments the selector to be '<branchtag>:Latest'. Appending ':Latest' to the selector might not match the work area selector. This mismatch invalidates your next incremental populate resulting in a slower, full populate.
- The -version option is mutually exclusive with the -mirror and -merge options unless you specify '-version Latest'. You do not have to specify -version with -mirror.
- The -version and -overlay options are mutually exclusive.

--

-- Indicates that the command should stop looking for command options. Use this option when an argument to the command begin with a hyphen (-).

RETURN VALUE

In dss/dssc mode, you cannot operate on return values, so the return value is irrelevant.

In stcl/stclc mode, two lists are returned, where the first list is non-empty if at least one object was successfully processed, and the second list is non-empty if at least one object failed.

```
Notes:
 - "successfully processed" may not mean "successfully checked
   out". For example, attempting to check out an object that you
   already have in your work area is considered a success even though
   no checkout occurs.
 - Scripts should only test for non-empty lists to
   determine success or failure. The actual content of a non-empty
   list currently takes the form "Objects succeeded (n)" and
   "Objects failed (n)", where 'n' is the number of objects that
   succeeded or failed. However, these return values are subject to
   change in a future release.
 - If all objects fail, an exception occurs (the return value is
   thrown, not returned).
 - If a comment editor is defined, but cannot be used, the command
   automatically switches to the interactive command-line comment
   mode.
```

SEE ALSO

```
ci, populate, cancel, localversion, select, setmirror, setselector,
selectors, command defaults
```

EXAMPLES

Example of Checking Out a File with a Lock

```
The following example checks out for editing the Latest version of
a single file. The branch is locked to prevent others from checking
in a newer version until after the changes are made and the file is
checked in, or the checkout is canceled.

  dss> co -lock -comment "Trying an experiment" pcimaster.vbh

After making changes to the file, you check in using the following
command, creating a new version:

  dss> ci -comment "Experimental version" -keep pcimaster.vbh
```

Example of Checking Out a File From a Branch

In the following example, the version of 'top.v' associated with the 'Gold' tag is fetched. If 'Gold' is a branch tag, then the Latest version of 'top.v' on that branch is fetched. Otherwise, the version tagged 'Gold' is fetched. If 'top.v' does not have a branch or version tagged 'Gold', the checkout fails.

```
dss> co -get -version Gold top.v
```

Example of Updating File Links in the Current Directory Recursively

In the following example, an entire work area is updated to have links to the Latest versions in the cache directory:

```
dss> co -rec -share *
```

Links are only created for those objects you already have in your work area. If your intent is to have your work area reflect the current status of the vault, including any new files that are not in your work area, use the populate command instead:

```
dss> pop -rec -share
```

Example of Merging a File into Your Work Area

The following example demonstrates the non-locking (merging) design methodology. The vault for 'alutest.txt' contains four versions: 1.1 through 1.4. Two developers, Ann and Bill, fetch (check out without a lock) version 1.4 of 'alutest.txt':

```
dss> co -get alutest.txt
```

Both Ann and Bill make modifications to their local copies, and Bill checks in his changes first:

```
dss> ci -nocomment -keep alutest.txt
```

The checkin succeeds, creating version 1.5 in the vault. Later, Ann tries to check in the file, but DesignSync errors because a new Latest version exists in the vault. An 'ls' command on 'alutest.txt' confirms the status of the file as 'Needs Merge'. Ann must merge the changes contained in the Latest version into her local copy:

```
dss> co -merge alutest.txt
```

DesignSync merges version 1.5 with Ann's local copy and informs Ann if there are any merge conflicts. Ann must resolve any conflicts before she can check in the merged file. (See the -merge option description for information on resolving conflicts.) Once conflicts are resolved, Ann's local file contains both her changes and Bill's changes, and Ann can now check in the file, creating

```
version 1.6:

  dss> ci -nocomment alutest.txt
```

Example of Merging From a Different Branch

```
In the following example, you are working on the main development
branch (Trunk), but need to pick up some bug fixes that were made
on another branch (Dev). Because you are merging from another
branch, you use the combination of -merge and -overlay:

  dss> co -merge -overlay Dev:Latest alu.v decoder.v

DesignSync fetches the Latest versions of 'alu.v' and 'decoder.v'
from the 'Dev' branch and merges them into your local copies from
the 'Trunk' branch.

If you wanted to pick up all the changes from another branch, you
would use the 'populate -merge -overlay' command.
```

Example of Checking out and Locking a File Reference

```
In the following example, you have a managed object that you intend
to regenerate.  You want to obtain a lock, but you do not want to
fetch the existing object as you intend to overwrite its contents.
You use the -lock and -reference options to create a locked
reference:

  dss> co -reference -lock -nocomment top.netlist
```

Example of Checking Out Objects With Different Version Selectors

```
This example shows the checkout of multiple objects with differing
version selector.
   stcl> co file1;1.2 file2;bugfix:Latest file3;rdy_for_testing

This will fetch version 1.2 of file2, the Latest version of file2
on the bugfix branch, and the version of file3 tagged
"rdy_for_testing" on the default Trunk ("1") branch.

You can also use wildcards. For example,
   stcl> co "f*;1.5"
will fetch version 1.5 of all object names beginning with "f*".

You cannot check out folders by specifying the version selector.
For example:
   stcl> co -rec "subdir;1.5
will fetch items into the "subdir" folder, ignoring the selector
"1.5".
```

Example of Checking Out a Collection Object

This example shows the checkout of a collection object that deletes local versions.

Note: The DesignSync Milkway integration has been deprecated. This example is meant to be used only as a reference.

Mike checks out the Milkyway collection object top_design.sync.mw, which fetches local version 4 of that object to his workspace. He modifies the object and creates local version 5. Then he checks in top_design.sync.mw. The check-in operation does not remove local versions, so Mike now has local version 5 (unmodified) and local version 4 in his workspace. (Note: Because the checkin removes local version 4's link to with the original check-out operation of top_design, DesignSync now considers local version 4 to be modified.)

Ben checks out top_design.sync.mw (local version 5). He creates local version 6 and checks the object in.

Mike does some work on top_design, which creates local versions 6, 7, and 8 in his workspace. Then he decides to use Ben's version of the top_design object instead.

Mike checks out the top_design object (local version 6) from the vault. He doesn't want to save his local versions of the object, so he uses the '-savelocal delete' option to delete local versions other than the local version being fetched. In addition, he uses the -force option. (Because he created local versions 6, 7, and 8 of top_design in his workspace, DesignSync considers the top_design object to be locally modified and by default the checkout operation will fail. To successfully check out top_design, Mike must use '-force'.)

```
stcl> cd /home/tjones/top_design_library
stcl> co top_design.sync.mw -savelocal delete -force
```

Before fetching top_design.sync.mw from the vault, the check-out operation first deletes all local versions that are unmodified. So the check-out operation deletes Mike's local version 6 because that was the version originally fetched and its files are unmodified.

Because Mike specified the -force option, the checkout also deletes Mike's local version 8 (the current local version containing modified data for the object).

Because Mike specified '-savelocal delete', the check-out operation deletes local version 7, which is not in the vault and is not the modified data Mike agreed to delete when he specified '-force'. If Mike specified '-savelocal save', DesignSync would save local version 7. Local version 4 is also deleted.

Finally, Mike's check-out operation fetches the top_design object (Ben's local version 6) from the vault.

```
Mike continues to modify the top_design object, creating local
version 7, which he checks in.

Ben has local versions 5 and 6 in his workspace. He checks out the
top_design collection object (local version 7), specifying
'-savelocal fail'. The check-out operation removes local version 6
from his workspace because it is unmodified. The operation saves
local version 5 even though it is modified. (Ben's checkin of local
version 6 removed local version 5's link to with the original
checkout of top_design, so DesignSync now considers local version 5
to be modified.) The checkout also takes place despite the fact that
Ben specified '-savelocal fail'. The check-out operation takes this
action because local version 5 has a number lower than the local
version being fetched. If Ben had instead specified 'co -savelocal
delete', the checkout would delete local version 5.
```

populate

populate Command

NAME

```
populate            - Fetches or updates specified objects
```

DESCRIPTION

- Object States
- How Populate Handles Selectors
- Populate Log
- How Populate Handles Collections with Local Versions
- Populating Module Objects (Module-based)
- Setting up Your Workspace (Module-based)
- How Populate Handles Module Snapshots (Module-based)
- How Populate Handles Module Views (Module-based)
- Resolving Module Conflicts with Populate (Module-based)
- Module Cache (Module-based)
- External Module Support (Module-based)
- Populating Modules Recursively (Module-based)
- Module Version Updating (Module-based)
- Incremental Versus Full Populate (Module-based)
- How Populate Handles Moved and Removed Module Members (Module-based)
- Merging Across Branches (Module-based)
- Understanding the Output (Module-based)
- Forcing, Replacing, and Non-Replacing Modes (Module-based)
- Interacting with Legacy Modules (Legacy-based)
- Incremental Versus Full Populate (Legacy-based)
- Setting up Your Workspace (File-based)
- Incremental Versus Full Populate (File-based)

```
This command fetches the specified objects from the server
into your current workspace folder or a folder you specify
with the -path option.

Typically, you create your work area, or workspace, and perform your
first populate, an initial populate, as a full populate.  Once your
work area is populated, you can use the populate, co, and ci commands
to selectively check out and check in specific objects. You should
also populate periodically to update your work area with newly
managed objects, as well as newer versions of objects you have
locally.

Populate is used to create or update the objects in your
workspace. Populate features many ways to control the data brought
into your workspace. Because of the complexity of the populate
features, the description section is divided into sections that
detail the major features and functionality of populate.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

This command supports the command defaults system.
```

Object States

```
Upon populating your workspace, DesignSync determines in what
state to leave the fetched objects in your work area:
  1. DesignSync obeys the state option (-get, -lock, -share,
     -mirror, -reference) specified on the command line.
  2. If no state option is specified, DesignSync uses the default
     fetch state as specified by your project leader. See
     the "fetch preference" help topic for more information.
  3. If a default fetch state is not defined, the default
     behavior for 'populate' is -get.

Important: For both incremental and full populate operations,
DesignSync changes the state of only those objects that need
updating.  DesignSync does not change the state of up-to-date
objects during the populate operation.

The following methods let you override the default behavior
to change the states of all objects during a populate operation:
  o To change the state of up-to-date objects during a populate,
    use the -unifystate option.  To change the state of all
    objects that need an update as well as up-to-date
```

and locally modified objects, use -unifystate with the -force option.
- Unlocked locally modified objects are not overwritten unless you specify -force. For example, if you modify a fetched file, then execute a 'populate -share' command, your locally modified file is not replaced by a link to a file in the cache unless you also specify -force. Locked files are not overwritten by the -force option.
- To make populating with links to the mirror a fast operation, links are created only if no object (locally modified or not) or link already exists in your work area. You must specify -unifystate to change the state of existing objects and links in this case. Use -force, as well, to overwrite locally modified objects that are not locked and to remove objects that are not in the current configuration.

Note: If the object is designated as uncachable, attempts to place objects in the cache (populate -mirror; populate -share) will automatically populates the workspace with unlocked copies (-keep mode). For more information on cachability, see the "caching" commands.

How Populate Handles Selectors

DesignSync determines what versions of objects to populate as follows:

1. DesignSync obeys the selector list specified by the -version option.
2. If -version is not specified, DesignSync uses the persistent selector list of the top-level folder being populated. The default persistent selector is 'Trunk', in which case DesignSync checks out the Latest versions from Trunk.

Notes:

- If you specify a selector or a selector list for the populate operation using the -version option and the selector does not exactly match the workspace selector, an incremental populate is no longer valid. In this case, DesignSync performs a full populate even if the -incremental option is specified. See "Incremental Versus Full Populate" above for more information.

Important: The persistent selector lists of individual managed objects (files or collections) and subfolders are not obeyed by the 'populate -recursive' operation.

- A 'populate -recursive' command without the -version option populates a work area based on the persistent selector list of the top-level folder you are populating, skipping any subfolder or managed object that has a persistent selector list that differs from the top-level folder. You must issue the populate command separately for any skipped subfolder.

- A 'populate -recursive -version <selectorList>' command uses the specified selector list and ignores all persistent selector lists. In the case of '-version Latest', the persistent selector list of the top-level folder being populated is augmented with 'Latest' and that augmented persistent selector list is used for the populate operation.

The supported DesignSync use models (single-branch development, project branching, and auto-branching) assume that persistent selector lists across a work area are consistent. Use caution when using commands that leave you with inconsistent local metadata, such as using 'setselector' or 'mkbranch' on individual objects.

See the "selectors" help topic for details on selectors, selector lists, and persistent selector lists. For more information about how the -version switch is managed, see the -version in OPTIONS.

Populate Log

Because populate operations can be long and complex, you may want to specify a log file to contain only the output of the populate command to store for later reference.

You can specify the log file on an as needed basis using the -log option or by setting a log file name using the command defaults system. If the log file specified does not exist, DesignSync creates it before it begins the populate command processing. If the log file does exist, DesignSync appends the new populate information to the file.

Tip: If you set a default log value for populate, check the file size periodically and, if the file is getting too large to use comfortably, rename the file to save the information, or remove the file if you no longer need it.

Notes:

- If a log file is defined in the command defaults system and two users run populate simultaneously, the populate output may become interlaced in the log file.

- Regardless of whether you create a populate log, the DesignSync client log file contains the output of the populate command along with all the other commands typed into the DesignSync client session.

How Populate Handles Collections with Local Versions

For collection objects that have local versions (for example, custom generic collections), the populate operation handles local versions in the following way.

When you populate a folder containing a collection object, the populate operation removes from your workspace any local version of the object that is unmodified. (Because these local versions exist in the vault, you can refetch them.) The operation then fetches from the vault the specified collection object (with the local version number it had at the time of checkin).

If the current local version in your workspace is modified, the populate operation fails unless you specify 'co -force'. (The -force option lets the local version with the modified data be replaced with the local version of the object you are checking out.) Note: The current local version is the one with the highest local version number. DesignSync considers a local version to be modified if it contains modified members or if it is not the local version originally fetched from the vault when the collection object was checked out or populated to your workspace.

The -savelocal option tells the populate operation what to do with local versions in your workspace other than a current local version that is modified. For information, see OPTIONS.

Populating Module Objects (Module-based)

The populate command recognizes and fetches hierarchical module structure. These modules are data that represent a level of the design hierarchy. Such data includes objects or an entire vault folder hierarchy of objects managed in DesignSync, as well as hierarchical references to other modules. These modules can be stored on other SyncServers. For more information about modules, see DesignSync Data Manager User's Guide: "What is a Module?".

Important: You must use the populate command rather than the co command when fetching modules or module objects. The co command does not support modules.

To specify a module for an initial populate, you must specify its server URL, in the following format:
sync://<machine>:<port>/Modules/<category>/<module_name>[;<selector>]

DesignSync looks for an existing workspace root. If no workspace root exists and the registry key AllowAutoRootCreation is enabled, DesignSync automatically creates the workspace root based on the value set for DefaultAutoRoot path. If there is no existing workspace root path and DesignSync cannot create one, the populate fails. Workspace root path settings are in the DesignSync registry.

During the initial populate, DesignSync performs an implicit setvault. If necessary, DesignSync also creates a workspace folder for the module. For subsequent populates, you do not have to specify the server URL for the module; you can populate the module by specifying just the module name or the module instance name if your current directory is within the workspace root (see the setroot command help), or using the full workspace address which is "<module base

directory>/<module instance name>".

If a top-level module (a module that is not hierarchically subordinate to another module populated in the workspace) is populated with the -version option, the persistent selector for the workspace is changed to the version specified.

Overlapping of modules is supported. You use the -modulecontext option to indicate which module to populate if more than one module exists in the current directory (or that specified with the -path option). If no -modulecontext option is specified, all appropriate module objects from the candidate modules are populated.

If a file is a member of both overlapping modules, a populate clash occurs. In this case, the first module to populate the file 'wins'. A subsequent attempt by an overlapping module to populate the same file fails.

Two different versions of the same module cannot share the same base directory. However, you can populate two versions of the same module side by side.

Notes:
- Mirrors are not supported with module objects; you get an error if you use the -mirror option.
- If a module member is checked out with a lock, the locker keyword is not expanded with the locker name.
- You can use the -mcachemode, -mcachepaths, or -noreplace options only when populating a directory that is part of a module or a legacy module.
- After the upgrade command has been used to convert legacy modules to a module, fetch each new module to an empty work area. The upgrade command does not upgrade existing work areas.

Setting up Your Workspace (Module-based)

Before you can use populate to maintain your workspace, you must set up your workspace.

Note: The Workspace Wizard from the DesignSync graphical user interface simplifies the task of setting up a work area by taking you through a step-by-step process.

The typical steps when you set up a new work area are:

1. Create the folder for your workspace, if it does not already exist.

2. Populate the work area with the specified design objects from the vault. Populate determines the set of versions from the persistent selector list or from the -version option, if applicable. Apply the -recursive option to create a local hierarchy that matches the vault's hierarchy. Without -recursive, populate only fetches the specified objects.

How Populate Handles Module Snapshots (Module-based)

A module snapshot is a set of meaningful tagged module objects. The content and structure of a module snapshot is frozen to preserve important configurations. After the module snapshot has been created using the tag command, you can populate the snapshot into a local workspace for viewing, testing, or integrating into other work.

When you populate a module snapshot as a fixed workspace for viewing or testing, you use the snapshot tag as a selector. This can be either the full snapshot branch and version name or the simple tag name. When you populate a snapshot module, you can update tags on module members or hrefs within your workspace, but cannot checkin any content or other structural changes to the module members or the module.

When you populate a module snapshot to integrate with other work, you populate using a comma separated list of selectors ending with a "main" selector. This populates from the main selector first and replaces any matching objects with the member objects from the selectors in the selector list.

This results in a workspace that uses the main selector as the base and the destination for any checkins, but some or all of the module member objects from the snapshot workspaces. For example, specifying the following version to populate:
Beta,Alpha,Trunk:Latest

The populate command creates a module manifest from the main selector, Trunk:Latest, and overlays that with the contents of the Alpha version, and then the Beta version. The final manifest is then sent to the client. The server uses the natural path of the objects and the uuid to determine which module members to replace.

When hierarchical references are populated as part of the operation, the hierarchical reference versions come from the main selector list, not from the specified module snapshots.

When the hierarchical references are populated recursively during the initial populate using a selector list, the module members within the populated submodules are also populated with the selector list. If hierarchical references are not populated recursively during the initial populate using a selector list, they will not overlay member items from the selector list on subsequent populates.

Notes:

- If the "main" selector list is a snapshot branch, or a static selector of any type, you will not be able to check in any changes from the workspace.
- When populating a selector list, the module member objects in the specified snapshot are populated instead of the objects in the main selector. Populate will never attempt to merge the members.

If you want to merge data from a module snapshot into your workspace, you will not use a selector list, but populate your snapshot with the -merge and -overlay options into a workspace that has the default selector defined as the desired destination for checkin.

- Any hierarchical references that are defined as a static module version indicated by the selector on the href will not inherit any the selector list, even if the initial populate specifies using the selector list recursively.

How Populate Handles Module Views (Module-based)

A module view is a defined subset of module members and hierarchical references that have significance as a unit. The module view definition is stored on the server with a unique module view name. During populate, you can specify the view name to restrict the populate operation to only those members in the view. You can populate using more than one view.

Note: During initial populate, if you specify a view, the view specified persists in the workspace.

The populate operation builds the list of module members and hierarchical references (if run recursively) to populate by first looking at the specified view(s) on the specified module and selector. After building this aggregate set of data, DesignSync applies the filtering rules from the -filter, -hreffilter and -exclude options to determine what objects to populate into the workspace.

On an initial populate, the module view name or names list provided is propagated through the hierarchy and applied to all fetched modules. The module view name or names list is also saved, or persisted in the workspace metadata so that all subsequent populates use the same view. The documentation refers to a view saved in the metadata as a "persistent module view" because, like a persistent selector, it persists through subsequent populates rather than needing to be specified with each command.

If a persistent module view has been set on a module instance in a workspace any sub-modules subsequently populated use the persistent module view already defined by default.

Note: You can set or clear a persistent selector by using the setview command.

Resolving Module Conflicts with Populate (Module-based)

DesignSync provides the ability to define an overriding hierarchical reference to be used in cases where submodule references point to different versions of the same object. This can be used in both a

peer-to-peer or hierarchical cone structure. In a peer-to-peer structure, it can be used to resolve conflicts and determine which version of the sub-module to populate into workspace.

For example, a module called TOP with hrefs to sub-modules:
ROM@1.23 -relpath ../ROM
COM@1.15 -relpath ../COM

where ROM and COM both contain an href to a common libraries directory, but to different versions:
ROM -> LIB@1.3 -relpath ../LIB
COM -> LIB@1.5 -relpath ../LIB

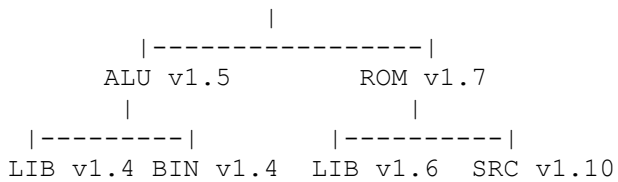
Working in a peer-based structure, where your modules are populated in a flat directory setting, your workspace may look something like this:
/home/workspace/TOP
/home/workspace/ROM
/home/workspace/COM
/home/workspace/LIB

DesignSync may experience a conflict determining what version of LIB (1.3 or 1.5, as referenced in the hierarchy)to populate in the peer directory /home/workspace/LIB.

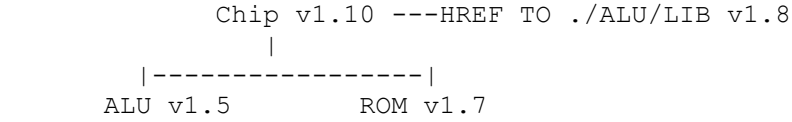
If an href is placed higher in the peer structure, however; it will become the overriding href. So, for example, if you add an href for TOP to LIB, as shown:
```
 TOP -> ROM@1.23 -relpath ../ROM
     -> COM@1.15 -relpath ../COM
     -> LIB@1.5 -relpath ../LIB
```

When you populate the TOP workspace recursively into /home/workspace/TOP, DesignSync populates the LIB directory with the 1.5 version, eliminating the guesswork.

In a cone structure, it can be used to substitute a submodule version without modifying the hierarchy or branching the sub-module to update an href version. For example:



If rather than branching ALU and updating the hierarchical reference to LIB, you add an href to the desired version of LIB at a higher level, for example, Chip, then that version of LIB will replace the lower level version with the same relpath when populated.



```
        |                  |
  |-----------|        |-----------|
LIB v1.8 BIN v1.4   LIB v1.6 SRC v1.10
```

Notes:

o The relpath of the hierarchical reference is what's used to determine which sub-module is replaced.

o In order for the overriding href to be used by the system, you must populate recursively from the highest level module containing the override href. For example, if you were to populate either of the above examples at the ROM level, the ROM href is the one that is used to determine what submodule is populated; not the higher-level module.

Module Cache (Module-based)

A module cache (mcache) can be thought of as a shared workspace. The populate command works with both module and legacy module mcaches. A module mcache contain modules while a legacy mcache contains only legacy releases.

To create a module cache, team leaders should create a workspace and populate it with modules and or legacy modules using the -share option. This becomes the mcache directory. Usually a team leader creates the mcache for team members to access over the LAN. The mcache should be writable only by the team leader. Team members should need permission to read the data, link to and copy the module or legacy module in the mcache.

Note: The module cache must be the workspace root directory.

An mcache is manually administrated. Modules and legacy modules can be fetched as needed. You can have multiple modules in the mcache.

o You can have full copies of all the modules in an mcache.
o If you use -share option to populate an mcache, it allows you to keep full copies of the DIFFERENCES between versions by populating the mcache from the DesignSync cache which stores the files.

Note: Only statically fetched modules can be fetched from an mcache during populate.
Only released configurations can be fetched from an mcache during populate.

Since multiple modules can have the same base directory or have the same directory at various levels, it can cause confusion for mcache links and can even cause circular or inconsistent links. To keep the contents of a mcache consistent, an mcache link to an mcached directory containing modules are created for only one module version.

An mcache can either be for modules or legacy modules, not both. A module can have hierarchical references to legacy modules, resulting

in the legacy modules being populated to the module mcache. These legacy modules are ignored when creating mcache links or copies.

The -mcachemode copy option is ignored for modules. You can, however, get the contents of a module from the LAN if your team lead fetches the modules from the server into the mcache using the -share option. This forces the module contents to get fetched into the DesignSync cache (different from an mcache). Symlinks are created in the mcache to point to these files in the DesignSync cache.

If you specify -mcachemode copy to get full copies of a module's contents from the mcache, the populate operation automatically switches the command to use the default '-from local' mode to fetch the files.

To use a module mcache the root directory of the mcache must be provided in the -mcachepaths option or the mcache paths registry setting. This root directory contains the metadata identifying the base directories of all module cache. See the section on -mcachepaths for more information.

Note: If a module, module category, the Module area or server is designated uncachable, it cannot be stored in an mcache. If a module has already been populated into a cache and is then designated as uncachable, the module cache is not automatically removed.

External Module Support (Module-based)

DesignSync supports populating an external module, an object or set of objects managed by a different change management system, within a module hierarchy. Using an external module in a DesignSync hierarchy allows you to manage code dependencies between module objects in DesignSync and files checked in to other change management systems.

Within a parent module, you add an href that refers to an external module. The external module reference contains the name of an external module interface. The external module interface, provided by an administrator, defines a procedure to populate the sub-module using an external change management system.

After creating the href to the external module, you populate it exactly as you would any other href, by specifying either the href name or the module instance name as the populate argument, or by populating the parent module with the -recursive option.

The external module must be part of a module hierarchy. You cannot create an external module as a top-level module. Once in the workspace, the module itself, or any subfolders, or objects within the module may be individually populated according to the external module interface definition.

Notes:
- The external module's directory structure cannot overlap with any other module data.

o If an external module populate fails and the populate command was run with the -report brief option specified, you may not have enough information to determine where the failure occurred. If you rerun the populate with the -report brief mode, you can locate the referenced object within the module hierarchy.

Populating Modules Recursively (Module-based)

You can use populate to fetch entire modules or their members as follows:

o To fetch a single module without fetching its submodules, specify the workspace or server module and apply the populate command without the -recursive option.
The command populates the module members without following hierarchical references (hrefs).

o To fetch all objects in an entire module hierarchy, specify the workspace or server module and use the populate command with the -recursive option.
The command traverses the hierarchy in a module-centric fashion, populating all the objects in the module and following its hrefs to populate its referenced submodules.

o To fetch all objects in a folder, specify a folder name and apply the populate command without the -recursive option.
The command fetches the objects in the folder, without following hrefs.

o To fetch all objects in a folder and its subfolders, specify a folder name and apply the populate command with the -recursive option.
The command traverses the folders in a folder-centric fashion, populating the modified objects in the folder and its subfolders, but without following hrefs. To follow hrefs, you must specify a workspace or server module instead of a folder.

o To fetch all objects in a module or module hierarchy but restrict the fetch to a particular folder hierarchy, use the -modulecontext option to specify the module and provide the folder name.
- Specify the -recursive option if the module hierarchy needs be traversed to fetch items from the sub-modules into that folder.
- Specify -norecursive option to fetch only the items from the given module. Note that this operation is "module centric" and "folder recursive", in that all items in the module are fetched which belong to the given module or its sub-folders.
- To restrict the operation to both a module and a single folder, use the -filter option to filter out items from sub folder.

Note: You cannot specify the -recursive option, if you are performing a cross-branch merge (with pop -merge -overlay) on a module.

When you fetch a module recursively, you update the module hierarchy.

How that module hierarchy populates depends on the href mode specified, and the selector(s) specified within the href, the hreffilter string and possibly the populate selector for the selected module. For more information on how the module hierarchy is populated, see the "Module Hierarchy" topic in the ENOVIA Synchronicity DesignSync Data Manager User's Guide.

Note: If the "HrefModeChangeWithTopStaticSelector" registry key is enabled, and the selected module is a static version, the static version is saved as the persistent selector in populate. For more information about setting the "HrefModeChangeWithTopStaticSelector" registry key, see the ENOVIA Synchronicity DesignSync Data Manager Administrator's Guide.

Module Version Updating (Module-based)

The populate command updates the module version upon successfully fetching all members of the module. If the populate command is not completely successful, the fetched version number is not updated, as in the following scenarios:

- A module member cannot be fetched if the member is locally modified (and -force is not applied). In this case, the module is not fully populated, and the module version is not updated.

- A module member is not fetched if a -filter, -exclude, or -nonew option excludes it. In this case, the module is not fully populated, and the version number is not updated.

If you do not have the Latest complete module version due to one of these cases, you can still check in a module; the ci command auto-merges members so that the module is fully updated upon checkin. See the ci command for details.

You can use the showstatus command to detect whether a module has been fully populated. The showstatus command lists the module as 'Needs Update' if the Latest version has not been successfully fetched.

Unlike the cases where the module version is not updated, the module version is updated if a populate successfully updates the entire module, but fails to remove files that are no longer members of the module. If a member has been removed from the new module version, but the populate command cannot remove it from the workspace (because it is locally modified and -force was not applied), the workspace does contain the entire contents of the new module version, so the module version is updated.

Incremental Versus Full Populate (Module-based)

By default, the populate command attempts to perform an incremental

populate which updates only those local objects whose corresponding vaults have changed. For modules, DesignSync tracks the members changed on the server and in the workspace and performs an incremental populate. Avoiding a full populate improves the speed of the populate; however, some circumstances make a full populate necessary. In the following cases, DesignSync automatically performs a full populate:

- If you are populating with a different configuration to that of the work area (having used setselector, setvault, 'populate -version', or 'co -version' to change a selector), DesignSync performs a full populate. For example, if your last full populate specified the VendorA_Mem configuration, but you now want VendorB_Mem files, then DesignSync automatically performs a full (nonincremental) populate. If the selector you specify resolves to the same exact selector as that of the work area, DesignSync does perform the incremental populate. In this case, the selectors must be an exact match; for example, a selector which resolves to 'Main' does not match 'Main:Latest'. If you are populating with a new configuration, consider using the -force option to remove objects of the previous configuration from your work area.

- If you have removed module data from the workspace with rmfile or rmfolder, DesignSync performs a full populate, refetching the removed files.

- If you use the -lock option, DesignSync performs a full populate.

- If you use the -unifystate option, DesignSync performs a full populate.

- If you perform a nonrecursive populate on a subfolder, all of the folders above the subfolder are invalidated for subsequent incremental populate operations. Incremental populate works by exploiting the fact that if a folder is up-to-date, all of its subfolders are also up-to-date, making it unnecessary to recurse into them. Because a recursive populate was not performed for the subfolder, DesignSync cannot ensure that its subfolders are up-to-date; thus, all incremental populates are invalidated up the hierarchy.

- If you perform a nonrecursive populate on a folder, DesignSync essentially runs a full populate rather than the default incremental populate. Your next populate is incremental from the last recursive populate. If you have not previously run a recursive populate, the subsequent populate is a full populate.

Note: If you are using a mirror (by specifying -mirror or having a default fetch state of Links to mirror), an incremental populate does not necessarily fetch new objects checked in, nor remove links to objects deleted by team members until after the mirror has been updated.

For the following cases, you should perform a full populate instead of an incremental populate:

- If you have excluded a folder by using the -exclude, -filter, or -noemptydirs option with the populate command, a subsequent incremental populate will not necessarily process the folder of the previously excluded object. DesignSync does not automatically perform a full populate in this case. To guarantee that previously excluded objects are fetched, specify the -full option for the subsequent populate operation.

- Specify a full populate to force data that has been manually removed, removed locally, or renamed locally to be fetched again from the server. If the file was renamed, you may have to specify the -force option as well.

  Also, specify a full populate if you have an unchanged, but out-of-date or out-of-sync version in your workspace to force DesignSync to fetch the up-to-date version of the object.

- If the system clock on the SyncServer machine where your vault is located is turned back (for example, to correct clock skew between machines), you must perform a full (nonincremental) populate to synchronize the client and server metadata.

- If you interrupt a populate operation (using Control-c, for example), you should use populate -full on your next populate of that folder.

The default populate mode is -incremental; however, your project leader can set this default using SyncAdmin.

If you are updating mirrors, use the -incremental option. If you specify the -full option, mirror updates can take a long time to complete.

Note: If you remove objects from the work area by using operating system commands rather than DesignSync commands, an incremental populate cannot fetch these objects. Perform a full populate (-full) or use the -unifystate option to fetch them.

How Populate Handles Moved and Removed Module Members (Module-based)

When you populate a module, DesignSync does not populate any module member that has been removed on the server. Existing module members in your local workspace that have been removed on the server are removed during a populate.

Module members that have been removed or moved locally, but those changes were not committed to server are preserved in the workspace unless populate is run with the -full and -force options which remove the local modifications (including the structural changes) and replace the workspace version with the server version.

Merging module members that have been removed or renamed is discussed in Merging Across Branches

Merging Across Branches (Module-based)

In multi-branch environments, you use the populate command to merge branches. In many cases, a new branch that is created is eventually merged back into the main development branch.

The branch being merged is populated into a workspace containing the destination branch using the populate command with the -merge and -overlay options. This type of merge is called "cross-branch merge."

As with all populate operations, cross-branch merging uses the filter and exclude filter lists set on the workspace, in the command defaults system,on the command line.

Note: Filtering on module workspaces is applied to the natural path of the module members. If a module member's natural path has changed, creating a situation where either the new location or the old location, but not both is excluded, the module member is included in the merge.

Important: When working with modules, you should lock your workspace branch before beginning a cross-branch merge. This reduces the risk of changes being committed by another user while you are merging the versions. After the merge has been completed, the changes have been reviewed and accepted, and the new module version created, unlock the branch to make it available for general use.

Merging includes two basic types of merging: file contents, and structural changes.

o File content merging:
  File content merging is applicable to all DesignSync objects including module members. DesignSync merges the contents of files with the same natural path to the best of its ability. If the files are binary files which cannot be merged, populate returns an error message.

o Structural change merging for Modules:
  Structural changes for modules are either committed when the module is checked in or can be individually committed. Structural changes for Modules include:

  - Removed objects - If an object is present in the local workspace, but has been removed on the merge version, it is marked with a metadata property to indicate that it was removed from the branch. If you want to remove it from the merged module version, you must manually remove the file from the workspace before creating the new module.

    If the object has been removed on the workspace, but:
    * is present on the server at the same member version removed

from the workspace, the object remains in the same state, and is removed from the server during the next checkin.

* is present on the server at a newer version or has been moved, or is on the overlay version, the new version is not merged into the workspace, and an error is returned stating there is new version. The version in the workspace remains in the removed state, but you will not able to check in the change until you resolve the merge conflict.

- Added objects - If an object is present in the merge version, but not in local workspace, it is added to the module and is checked into the module when the next checkin operation on the module or the module member is performed.

- Moved or Renamed objects - A moved (or renamed) object has a different natural path. Objects that have been moved on either the server or checked in from the workspace have been moved on the server. Objects that have been moved in the workspace, but have not been checked in are considered moved locally.

  If an object has been moved on the server, but not locally, the module member in the workspace retains the same name or location in the workspace, and a metadata property is added to the object to indicate the new path name. To determine what files have been moved, review the populate status information, log file, or run the ls command with the -merge rename option.

  If an object has been moved locally, and:

  * has been moved on the server to the same location, the merge operation is performed on the merged local version. Subsequent checkin checks in the merged file to the new location. If the content has changed, DesignSync will perform a content merge as well.

  * has been removed on the server, the new version is not merged into the workspace, and an error is returned by populate. new version. The version in the workspace remains in the moved state, but you will not able to check in the change until you resolve the merge conflict.

  * has been updated on the server, content changes are merged into the moved file, and subsequent checkin of the member moves the file on the server and updates the content.

  * has been moved on the server to a different location and updated, the content is merged, the workspace version remains in the same location in the workspace, and an error is logged in populate to alert you that the file has been moved on the server. In order to checkin, you must resolve the merge name conflict or checkin with the -skip option to move the file to name of the file in your local workspace.

  * and exists on the overlay version, the overlay version is not copied into the workspace, but a metadata property is placed on the local version to indicate that natural path of the object

```
      is different.  You can see a list of these differences by using
      ls -merged.

    Note: If a file marked as renamed is subsequently renamed again,
    or removed from the module, the metadata property indicating that
    the file was renamed by merge may persist.  To clear the
    property, perform the mvmember or remove command on the workspace
    object, or manually clear the property using the url rmprop
    command.

  - Added or Removed hierarchical references - Hierarchical reference
    changes cannot be merged.  You must manually adjust your
    hierarchical references.

After a cross-branch merge has been performed, you can view the
status of the affected files using the ls command with the -merged
<state> -report D options.  The -merged option allows you to restrict
the list to a particular type of merge operation (add, remove,
rename, all) and the -report D option shows you the current state of
the object in your workspace.  For more information, see the ls
command help.

When a merge is performed on a DesignSync object, a merge edge is
created automatically to maintain a list of the changes incorporated
by the merge.  This identifies a closer-common ancestor to provide
for quicker subsequent merges. When performing a cross-branch merge
on a module, however, you need to manually create the merge edge
after committing the selected changes.  For more information on
creating a merge edge, see the mkedge command.

For more information about merging, see the -merge and -overlay
options, and the DesignSync Data Manager User's Guide topic: "What Is
Merging?"

Notes:
o Auto-branching is not supported for modules; you cannot specify
  the auto-branching construct, auto(), for modules.
```

Understanding the Output (Module-based)

```
The populate command provides the option to specify the level of
information the command outputs during processing. The -report option
allows you to specify what type of information is displayed:

If you run the command with the -report brief option, the populate
command outputs a small amount of status information including, but
not limited to:
 o Failure messages.
 o Warning messages.
 o Version of each module processed as a result of a recursive
    populate.
 o Removal message for any hierarchical reference. removed as part of
   a recursive module populate.
 o Informational messages concerning the status of the populate
```

```
 o Success/failure/skip status

If you do not specify a value, or the command with the -normal
option, the populate command outputs all the information presented
with -report brief and the following additional information:
 o Informational messages for objects that are updated by the
    populate operation.
 o Messages for objects excluded from the operation (due to exclusion
   filters or explicit exclusions).
 o For module data, also outputs information about all objects that
   are fetched.

If you run the command with the -report verbose option, the populate
command outputs all the information presented with -report normal and
the following additional information:
 o Informational message for every object examined but not updated.
 o For module data, also outputs information about all objects that
   are filtered.
 o For module versions that have been swapped, output indicates when
   the selector of a swapped sub-module is being used.

If you run the command with the -report error option, the populate
command outputs the following information:
 o Failure messages.
 o Warning messages.
 o Success/failure/skip status messages.

Note: References to DesignSync Vault, IPGear Deliverables, or
External modules do not have a module instance name to add to the
object path. When running with the error report mode, if an object
within a referenced DesignSync Vault, IPGear Deliverable, or External
module fails, you may need to rerun the operation with the report
-brief option to locate the referenced object within the module
hierarchy.
```

Forcing, Replacing, and Non-Replacing Modes (Module-based)

```
You can use these three modes to specify how the populate command
updates your work area:

  o Forcing mode (specified with the -force option) synchronizes
    your work area with the incoming data, including locally
    modified objects.  In this mode, the populate command updates
    the managed objects in your work area. It replaces or removes
    managed objects regardless of whether the objects have been
    locally modified and whether they are members of the module
    being fetched. Thus, forcing modifies your work area
    to match the set of module members being fetched. Note: The
    default -noforce option operates as if -replace has been
    specified.

  o Replacing mode (specified with the -replace option) also
    synchronizes your work area with the incoming data, but without
    affecting locally modified objects (the default behavior).
```

For modules, the populate command updates managed members of the module that have not been locally modified. It also removes any unmodified managed objects that are not members of the module being fetched.

Replacing mode, unlike forcing mode, leaves intact managed objects that have been locally modified.

- Non-replacing mode (specified with the -noreplace option) is the least disruptive mode; this mode might require you to clean up the resulting work area data.

  In this mode, the populate command takes the incoming data and overlays it on top of the existing work area's data. It leaves intact both managed objects with local modifications and managed objects that are not members of the module being fetched. Thus, the work area essentially becomes a union of the data from the previous version and that of the module being fetched.

  Non-replacing mode, unlike forcing mode, leaves intact any objects that have been locally modified, and, unlike the replacing mode, leaves unmodified managed objects intact. See the -[no]replace option below for more details.

Notes:
- Unmanaged objects in your work area are not affected by any of these modes.
- The following are illegal combinations of options: -replace and -noforce, as well as inverse options, such as -replace and -noreplace.

Interacting with Legacy Modules (Legacy-based)

The general functionality provided by populate is provided for legacy modules by the hcm get command. The sections within populate that are specifically tagged for legacy modules refer to interactions with modules or files-based objects, when populate is used, or if populate is used on individual objects, not an entire legacy module configuration. For more information on updating legacy modules in your workspace, see the hcm get command.

Important: Legacy modules are modules generated prior to Developer Suite 5.0. The modern modules functionality provides significant improvements. You can update your legacy modules using the upgrade command.

Prior to Developer Suite 5.0, legacy modules were managed with module configurations. Modules no longer require "configurations". A configuration was a set of object versions sharing a common tag (for example, files of a version tagged 'Rel2.0' comprise the Release 2.0 configuration).

In ProjectSync, a configuration represents a state in the life-cycle

of a project. It has an owner, team members. When associated with a DesignSync vault, the configuration has a selector list (typically a tag) identifying the versions of DesignSync data that are part of the configuration.
ProjectSync project and configuration information is stored in a sync_project.txt file that is located in the project folder.

When you populate based on a name that corresponds to a ProjectSync configuration, DesignSync uses the selector list (typically a tag name) associated with that ProjectSync configuration to identify the versions to be populated. This scenario is called configuration mapping.

Configuration mapping is used when a configuration name does not have the same meaning for all modules of a project. For example, a project's Alpha configuration may consist of the Gold configuration of one module, the Rel20 configuration of another, and several other modules whose design files are actually tagged Alpha. Configuration mapping lets you identify these different versions of design data with one configuration name.

When you populate a configuration-mapped folder (either directly or through a recursive populate operation) and the selector you specify is mapped, the persistent selector list for that folder is set to the mapped value. For example, if the specified selector 'Alpha' is a configuration that maps to the 'Gold' tag, then the persistent selector list for that folder is set to 'Gold'. Further, if the folder references a different vault (as identified by the REFERENCE keyword in the sync_project.txt file) and you are doing a recursive populate, the persistent selector list for any subfolder is also set to the mapped value.

Notes:
- The case where a ProjectSync configuration and its associated DesignSync tag have the same name is not considered configuration mapping; the persistent selector list is not modified by the populate operation.
- Only the populate command (not co, ci, and so on) resolves the selector you specify to a ProjectSync configuration, if one exists.
- DesignSync does not follow chained configuration maps. For example, if the same sync_project.txt file has a configuration A mapped to tag B and a configuration B mapped to tag C, DesignSync does not map A to C. Unexpected behavior can result. To avoid chained configuration maps, consider using separate naming conventions for configurations and tags.

- If an legacy module populate fails and the populate command was run with the -report brief option specified, you may not have enough information to determine where the failure occurred. If you rerun the populate with the -report brief mode, you will can locate the referenced object within the module hierarchy.

For information on how populate works on a legacy module or an href to a legacy module, see the description of -recursive option. See ProjectSync User's Guide for more information on ProjectSync projects and configurations. See the "Working with Legacy Modules" book in

DesignSync Data Manager User's Guide for more information about legacy modules.

Incremental Versus Full Populate (Legacy-based)

By default, the populate command attempts to perform an incremental populate which updates only those local objects whose corresponding vaults have changed. Avoiding a full populate improves the speed of the populate; however, some circumstances make a full populate necessary. In the following cases, DesignSync automatically performs a full populate:

- If you are populating with a different configuration to that of the work area (having used setselector, setvault, 'populate -version', or 'co -version' to change a selector), DesignSync performs a full populate. For example, if your last full populate specified the VendorA_Mem configuration, but you now want VendorB_Mem files, then DesignSync automatically performs a full (nonincremental) populate. If the selector you specify resolves to the same exact selector as that of the work area, DesignSync does perform the incremental populate. In this case, the selectors must be an exact match; for example, a selector which resolves to 'Main' does not match 'Main:Latest'. If you are populating with a new configuration, consider using the -force option to remove objects of the previous configuration from your work area.

- If you use the -lock option, DesignSync performs a full populate.

- If you use the -unifystate option, DesignSync performs a full populate.

- If you perform a nonrecursive populate on a subfolder, all of the folders above the subfolder are invalidated for subsequent incremental populate operations. Incremental populate works by exploiting the fact that if a folder is up-to-date, all of its subfolders are also up-to-date, making it unnecessary to recurse into them. Because a recursive populate was not performed for the subfolder, DesignSync cannot ensure that its subfolders are up-to-date; thus, all incremental populates are invalidated up the hierarchy.

- If you perform a nonrecursive populate on a folder, DesignSync essentially runs a full populate rather than the default incremental populate. Your next populate is incremental from the last recursive populate. If you have not previously run a recursive populate, the subsequent populate is a full populate.

- If a DesignSync REFERENCE resolves to a different selector than that of the work area from which the populate command is invoked, DesignSync performs a full populate of the REFERENCED objects

rather than an incremental populate. DesignSync compares the -version selector with the work area configuration rather than the mapped configuration, so do not use the -version selector to specify a mapped configuration. Instead, if you suspect the configuration map file has been updated, use the -version selector to remap the configuration by specifying the original selector. DesignSync then performs a full populate and follows the updated REFERENCEs.

- If the ProjectSync configuration file, sync_project.txt, has been updated through the ProjectSync interface (Project->Edit or Project->Configuration), thus updating the DesignSync REFERENCEs, DesignSync performs a full populate. If, however, the configuration in the sync_project.txt file is hand-edited and not updated using ProjectSync, you must specify the -full option to force a full populate.

Note: If you are using a mirror (by specifying -mirror or having a default fetch state of Links to mirror), an incremental populate does not necessarily fetch new objects checked in, nor remove links to objects deleted by team members until after the mirror has been updated.

For the following cases, perform a full populate instead of an incremental populate:

- If you have excluded a folder by using the -exclude or or -noemptydirs option with the populate command, a subsequent incremental populate will not necessarily process the folder of the previously excluded object. DesignSync does not automatically perform a full populate in this case. To guarantee that previously excluded objects are fetched, specify the -full option for the subsequent populate operation.

- If the ProjectSync configuration file, sync_project.txt, has been hand-edited, thus updating the legacy module REFERENCEs, use the -full option to perform a full populate. If, however, the sync_project.txt file has been changed through the ProjectSync interface (Project->Edit or Project->Configuration), DesignSync performs the full populate without your having to specify -full. For more information, see "Interaction with Legacy Modules" below.

- If the system clock on the SyncServer machine where your vault is located is turned back (for example, to correct clock skew between machines), you must perform a full (nonincremental) populate to synchronize the client and server metadata.

- If you interrupt a populate operation (using Control-c, for example), you should use populate -full on your next populate of that folder.

The default populate mode is -incremental; however, your project leader can set this default using SyncAdmin.

If you are updating mirrors, use the -incremental option. If you specify the -full option, mirror updates can take a long time

to complete.

Note: If you remove objects from the work area by using operating system commands rather than DesignSync commands, an incremental populate cannot fetch these objects. Perform a full populate or use the -unifystate or to fetch them.

Setting up Your Workspace (File-based)

Before you can use populate to maintain your workspace, you must set up your workspace.

Note: The Workspace Wizard from the DesignSync graphical user interface simplifies the task of setting up a work area by taking you through a step-by-step process.

The typical steps when you set up a new workspace are:

1. Associate a local folder with a vault folder. See the setvault command for details. This also creates the workspace root, if one does not already exist at the level of the local folder or above.

2. Optionally set the persistent selector list for the folder as part of the setvault command or with the setselector command. If you do not set the persistent selector list, it is inherited from the parent folder. This step is necessary only if you are working on a branch other than the default Trunk branch.

3. Optionally associate a local folder with a mirror directory. See the "setmirror" command for details. If the mirror directory for your project later changes, run the setmirror command from the same directory in which the original setmirror command was run. That will update the workspace's mirror association, which will be inherited by lower level directories. Run the populate command with the options '-recursive -mirror -unifystate' to correct existing workspace links to mirror files. This will correct the links so that they point to the mirror directory's new location.

4. Populate the work area with the specified design objects from the vault. Populate determines the set of versions from the persistent selector list or from the -version option, if applicable. Apply the -recursive option to create a local hierarchy that matches the vault's hierarchy. Without -recursive, populate only fetches the specified objects.

Incremental Versus Full Populate (File-based)

By default, the populate command attempts to perform an incremental populate which updates only those local objects whose corresponding

vaults have changed. Avoiding a full populate improves the speed of the populate; however, some circumstances make a full populate necessary. In the following cases, DesignSync automatically performs a full populate:

- If you are populating with a different configuration to that of the work area (having used setselector, setvault, 'populate -version', or 'co -version' to change a selector), DesignSync performs a full populate. For example, if your last full populate specified the VendorA_Mem configuration, but you now want VendorB_Mem files, then DesignSync automatically performs a full (nonincremental) populate. If the selector you specify resolves to the same exact selector as that of the work area, DesignSync does perform the incremental populate. In this case, the selectors must be an exact match; for example, a selector which resolves to 'Main' does not match 'Main:Latest'. If you are populating with a new configuration, consider using the -force option to remove objects of the previous configuration from your work area.

- If you use the -lock option, DesignSync performs a full populate.

- If you use the -unifystate option, DesignSync performs a full populate.

- If you perform a nonrecursive populate on a subfolder, all of the folders above the subfolder are invalidated for subsequent incremental populate operations. Incremental populate works by exploiting the fact that if a folder is up-to-date, all of its subfolders are also up-to-date, making it unnecessary to recurse into them. Because a recursive populate was not performed for the subfolder, DesignSync cannot ensure that its subfolders are up-to-date; thus, all incremental populates are invalidated up the hierarchy.

- If you perform a nonrecursive populate on a folder, DesignSync essentially runs a full populate rather than the default incremental populate. Your next populate is incremental from the last recursive populate. If you have not previously run a recursive populate, the subsequent populate is a full populate.

- If the ProjectSync configuration file, sync_project.txt, has been updated through the ProjectSync interface (Project->Edit or Project->Configuration), thus updating the DesignSync REFERENCEs, DesignSync performs a full populate. If, however, the configuration in the sync_project.txt file is hand-edited and not updated using ProjectSync, you must specify the -full option to force a full populate.

Note: If you are using a mirror (by specifying -mirror or having a default fetch state of Links to mirror), an incremental populate does not necessarily fetch new objects checked in, nor remove links to objects deleted by team members until after the

```
mirror has been updated.

For the following cases, perform a full populate instead of
an incremental populate:

  o If you have excluded a folder by using the -exclude, or
    -noemptydirs option with the populate command, a
    subsequent incremental populate will not necessarily process
    the folder of the previously excluded object.  DesignSync does
    not automatically perform a full populate in this case. To
    guarantee that previously excluded objects are fetched,
    specify the -full option for the subsequent populate operation.

  o For modules, DesignSync tracks changed members and therefore
    always performs an incremental populate. Specify a full populate
    to force data that has been manually removed, removed locally, or
    renamed locally to be fetched again from the server. If the file
    was renamed, you may have to specify the -force option as well.

    Also, specify a full populate if you have an unchanged,
    but out-of-date or out-of-sync version in your workspace to force
    DesignSync to fetch the up-to-date version of the object.

  o If the ProjectSync configuration file, sync_project.txt,
    has been hand-edited, thus updating the legacy module REFERENCEs,
    use the -full option to perform a full populate.  If, however,
    the sync_project.txt file has been changed through the
    ProjectSync interface (Project->Edit or Project->Configuration),
    DesignSync performs the full populate without your having to
    specify -full.  For more information, see "Interaction with
    Legacy Modules" below.

  o If the system clock on the SyncServer machine where your vault
    is located is turned back (for example, to correct clock skew
    between machines), you must perform a full (nonincremental)
    populate to synchronize the client and server metadata.

  o If you interrupt a populate operation (using Control-c, for
    example), you should use populate -full on your next populate
    of that folder.

The default populate mode is -incremental; however, your project
leader can set this default using SyncAdmin.

If you are updating mirrors, use the -incremental option. If you
specify the -full option, mirror updates can take a long time
to complete.

Note: If you remove objects from the work area by using operating
   system commands rather than DesignSync commands, an incremental
   populate cannot fetch these objects. Use the -unifystate or
   -full option to fetch them.
```

How Populate Handles Retired Objects (File-based)

When you populate with the Latest versions of design objects from a given branch, DesignSync does not populate objects for which that branch is retired. Objects in your local work area whose branches have been retired from the vault are not deleted during the populate operation unless you specify -force.

It is important to note that objects on retired branches remain part of past configurations. When you use the populate command to retrieve a configuration other than 'Latest', objects from retired branches are fetched. The populate command fetches objects from retired branches, thereby preserving past configurations, if the selector used for the operation is any of the following:

- A version tag other than 'Latest', even if the version tag points to the Latest version
- A version number, even if that number corresponds to the Latest version
- <branchtag>:Date(<date>) or <branchtag>:VaultDate(<date>)

Note: If the selector specifies a branch in the form '<branchtag>:', DesignSync augments the selector to be <branchtag>:Latest, meaning, 'Get the Latest version from the specified branch'. In this case, objects from retired branches are not fetched.

Note: For information about how retired files by cross-branch merge operations, see "Merging Across Branches."

Merging Across Branches (File-based)

In multi-branch environments, you use the populate command to merge branches. In many cases, a new branch that is created is eventually merged back into the main development branch.

The branch being merged is populated into a workspace containing the destination branch using the populate command with the -merge and -overlay options. This type of merge is called "cross-branch merge."

As with all populate operations, cross-branch merging uses the filter and exclude filter lists set on the workspace, in the command defaults system,on the command line.

Merging includes two basic types of merging: file contents, and structural changes.

- File content merging:
  File content merging is applicable to all DesignSync objects. DesignSync merges the contents of files with the same natural path to the best of its ability. If the files are binary files which cannot be merged, populate returns an error message.

- Structural changes for DesignSync objects.
  Structural changes for DesignSync objects are non-content based changes to the DesignSync objects that can affect the merge

results.

- Removed objects: If an object is present in the local workspace, but not in the merge version, the object in the local workspace is unchanged. If you want to remove it from the merged version, you must explicitly remove or retire the object.

- Added objects: If an object is not present in the local workspace, but is present in the merge version, the object is added to the local workspace. The merge action sets the following local metadata properties:

  - The current version is set to the fetched version, providing a meaningful branch-point version when you check the object into branch A.
  - The current branch information is undefined.
  - The persistent selector list for the object may be augmented to ensure that branch A is automatically created when you check in the object, thus eliminating the need to use ci -new. The following list explains how the persistent selector list is handled by the operation.
    1. If the first selector in the persistent selector list is a VaultDate() or Auto() selector, then the persistent selector list is not modified.
    2. If the first selector is of the form <branch>:<version>, then the first selector is modified to be Auto(<branch>).
    3. Otherwise, the first selector is modified to be Auto(<selector>). The object may be automatically checked in to the DesignSync vault, depending on the value of the persistent selector.

- Retired objects:

  - If the object is active in the workspace and retired on the branch version, the workspace version is unchanged.
  - If the object is retired or does not exist in the workspace, and is retired or does not exist on the branch, the workspace version is unchanged.
  - If the object is retired in the workspace and active on the branch version, the version from the branch version is merged with the workspace version. The object remains retired and must be unretired in order to be checked in.

After a cross-branch merge has been performed, you can view the status of the affected files using the ls command with the -merged <state> -report D options. The -merged option allows you to restrict the list to a particular type of merge operation (add, remove, rename, all) and the -report D option shows you the current state of the object in your workspace. For more information, see the ls command help.

When a merge is performed on a DesignSync object, a merge edge is created automatically to maintain a list of the changes incorporated by the merge. This identifies a closer-common ancestor to provide for quicker subsequent merges.

For more information about merging, see the -merge and -overlay options, and the DesignSync Data Manager User's Guide topic: "What Is Merging?"

Populate Versus Checkout (File-based)

The co and populate commands are similar in that they retrieve versions of objects from their vaults and place them in your work area. They differ in several ways, most notably:

- You typically use the co command to operate on objects that you already have locally, whereas populate updates your work area to reflect the status of the vault.
- The co command considers the persistent selector list for each object that is checked out, whereas populate only considers the persistent selector list for the folder that is being populated.

Note: The co and populate commands are gradually being merged.

Understanding the Output (File-based)

The populate command provides the option to specify the level of information the command outputs during processing. The -report option allows you to specify what type of information is displayed:

If you run the command with the -report brief option, the populate command outputs a small amount of status information including, but not limited to:

- Failure messages.
- Warning messages.
- Informational messages concerning the status of the populate
- Success/failure/skip status

If you do not specify a value, or the command with the -normal option, the populate command outputs all the information presented with -report brief and the following additional information:

- Informational messages for objects that are updated by the populate operation.
- Messages for objects excluded from the operation (due to exclusion filters or explicit exclusions).

If you run the command with the -report verbose option, the populate command outputs all the information presented with -report normal and the following additional information:

- Informational message for every object examined but not updated.

If you run the command with the -report error option, the populate command outputs the following information:

- Failure messages.
- Warning messages.
- Success/failure/skip status messages.

Forcing, Replacing, and Non-Replacing Modes (File-based)

You can use these three modes to specify how the populate command updates your work area:

- Forcing mode (specified with the -force option) synchronizes your work area with the incoming data, including locally modified objects. In this mode, the populate command updates the managed objects in your work area. It replaces or removes managed objects regardless of whether the objects have been locally modified. Thus, forcing modifies your work area to match the set of objects being fetched. Note: The default -noforce option operates as if -replace has been specified.

- Replacing mode (specified with the -replace option) also synchronizes your work area with the incoming data, but without affecting locally modified objects (the default behavior).

  Note: Retired files that have been kept or re-added to the workspace are considered locally modified.

  Replacing mode, unlike forcing mode, leaves intact managed objects that have been locally modified.

- Non-replacing mode (specified with the -noreplace option) is the least disruptive mode; this mode might require you to clean up the resulting work area data.

  In this mode, the populate command takes the incoming data and overlays it on top of the existing work area's data. It leaves intact both managed objects with local modifications and managed objects that are not members of the module being fetched. Thus, the work area essentially becomes a union of the data from the previous version and that of the module being fetched.

  Non-replacing mode, unlike forcing mode, leaves intact any objects that have been locally modified, and, unlike the replacing mode, leaves unmodified managed objects intact. See the -[no]replace option below for more details.

Notes:
- Unmanaged objects in your work area are not affected by any of these modes.
- The following are illegal combinations of options: -replace and -noforce, as well as inverse options, such as -replace and -noreplace.

SYNOPSIS

```
populate [-[no]connectinstances] [-[no]emptydirs]
         [-exclude <object>[,<object>...]] [-filter <string>]
```

```
[-[no]force] [-full | -incremental] [-hreffilter <string>]
[-hrefmode {static | dynamic | normal}]
[[-lock [-keys <mode> | -from {local | vault}]] |
[-get [-keys <mode> | -from {local | vault}]]
[-share] | [-mirror] | [-reference] [-lock -reference] ]
[-log <filename>]  [-mcachemode <mcache_mode>]
[-mcachepaths <path_list>] [-[no]merge]
[-modulecontext <context>] [-[no]new]]
[[-overlay <selector>[,<selector>...]]|
[-version <selector>[,<selector>...]]] [-path <path>]
[-[no]recursive] [-[no]replace]
[-report {error|brief|normal|verbose}] [-[no]retain]
[-savelocal <value>] [-target <module_configuration_url>]
[-trigarg <arg>] [-[no]unifystate] [-view view1[,view2,...]]
[-xtras <list>] [--] [<argument> [<argument>...]]
```

ARGUMENTS

- Server Module URL (Module-based)
- Workspace Module (Module-based)
- Module Folder (Module-based)
- Module Member (Module-based)
- Hierarchical Reference (Module-based)
- External Module (Module-based)
- DesignSync Object (File-based)
- DesignSync Folder (File-based)

```
The populate command accepts multiple arguments. If you want
to populate the current folder, you need not specify an
argument.  Otherwise, specify one or more of the following
arguments:
```

Server Module URL (Module-based)

```
<server module>     Fetches the specified module from its vault.
                    For an initial populate of a module, you must
                    specify the module's server URL in the format:
                    sync://<machine>:<port>/Modules/<category>/
                    <module_name>[;<selector>].

                    The populate fetches all objects within the
                    module, but does not follow hierarchical
                    references (hrefs) by default; specify the
                    -recursive option to follow hrefs and thus
                    fetch the module's submodules.
```

Workspace Module (Module-based)

<workspace module> Fetches the specified module from its vault, or updates the module to the appropriate module version specified by the selector in use.

The populate fetches all objects within the module, but does not follow hierarchical references (hrefs) by default; specify the -recursive option to follow hrefs and thus fetch the module's submodules.

Module Folder (Module-based)

<module folder> Populates objects in the specified folder regardless of which module the files belong to. Specify the -recursive option to recurse within the specified folder. Populate in this case, does not follow hierarchical references (hrefs).

Note: To populate a module folder, the folder must already exist in the workspace.

If you specify the -modulecontext option, the populate command updates the items belonging to the specified module in the specified folder and all the sub-folders. If you use the -recursive option in addition to the -modulecontext option, populate fetches any items from relevant sub-modules that fall within the folder specified (or its sub-folders.)

Specify the module folder as an absolute path or a relative path. If you specify a relative path, it is assumed to be relative to the current directory or that specified by the -path option.

Note: In previous releases, if the directory that was being populated was part of a legacy module, the entire module and not just the module members in the directory got populated.

Module Member (Module-based)

<module member> Fetches the module member.
You can specify the -modulecontext option if more than one module exists in the workspace.

Note: The -modulecontext option is not normally needed, as the system knows what module each member belongs to. When there are

multiple overlapping modules and you are fetching an object that is not currently in the workspace (for example, to fetch something that was originally filtered, or was removed with rmfile), the -modulecontext option can be used to identify the module from which the object should be fetched.

You can also provide the version-extended name if necessary. A version-extended name is a filename followed by a semicolon and a version number or tag name (for example, top.v;1.2 or top.v;rel13). In this case, DesignSync fetches the specific version of the member vault instead of fetching the version of this object that belongs with the module version.
Note: If you specify the version-extended name, populate ignores the -version option.

Hierarchical Reference (Module-based)

<href>	Fetches the referenced target (submodule) identified by the hierarchical reference (href). You can use -hreffilter to exclude submodules. To include submodules, enter the href as the argument of the populate command. To indicate the module context of the href, use the -modulecontext option. Note: You can only specify hrefs directly within the specified module. For example, if a module Chip has an href to module CPU, and module CPU has an href to module ALU, you cannot reference the ALU. Thus, the following command invocations are invalid: 'populate -modulecontext Chip ALU' and 'populate -modulecontext Chip CPU/ALU'.

External Module (Module-based)

<external module>	Specifies the module instance of the external module in the workspace or the URL of the external module version to which you wish to create the connection. An external module is an object or set of objects managed by a different code management system but available for viewing and integration through DesignSync. Specify the external module in the workspace as a module instance name. Specify the external module Server URL as follows:

```
sync[s]:///ExternalModule/<external-type>/<external-data>
       where ExternalModule is a constant that identifies
       this URL as an external module URL,
       <external-type> is the name of the external module
       procedure, and <external-data> contains the
       parameters and options to pass to the
       procedure. These parameters and options can be
       passed from the procedure to the external code
       management system or to DesignSync.

       Note: In order to specify an external module, you
      must have previously populated the module in the
      workspace. You may also specify the external
      module by href name only.
```

DesignSync Object (File-based)

```
<DesignSync object> Fetches the object from its vault.
```

DesignSync Folder (File-based)

```
<DesignSync folder> Fetches the contents of the specified folder.
                    You can also use the -path option to specify
                    a folder to be fetched.
```

OPTIONS

-[no]connectinstances (Module-based)

```
-[no]connectinstances   This option determines how to handle updating
                        hierarchical reference within a top-level
                       module.

                        If your workspace is set up
                        in a peer structure, containing your
                        top-level module and modules which are
                        referenced submodules, but have been
                        populated independently, then when
                        your workspace is populated non-recursively,
                        DesignSync does not recognize the connection
                        between the modules.  When populated
                        recursively, DesignSync may change the
```

selector of the submodules to match the hierarchical reference definition. The -connectinstances option allows you to populate the top-level module, recognizes that the peer modules are, in fact, referenced submodules, and creates the relationship accordingly, but does not update the selector to match the hierarchical reference definition.

This option is mutually exclusive with -recursive which updates the href to the referenced peer module.

The -noconnectinstances option does not establish or identify a hierarchical relationship with referenced peer modules. (Default)

Notes:
* You can use the -connectinstances option with the -hreffilter option to identify specific submodules instead of updating the relationships for the entire module hierarchy.

* The submodule must match the target module and relative path specified in the hierarchical reference in order to the update the href.

-[no]emptydirs

-[no]emptydirs

Determines whether empty directories are removed or retained when populating a directory. Specify -noemptydirs to remove empty directories or -emptydirs to retain them. The default for the populate operation is -noemptydirs.

For example, if you are creating a directory structure to use as a template at the start of a project, you may want your team to populate the empty directories to retain the directory structure. In this case, you would specify 'populate -rec -emptydirs'.

If a populate operation using -noemptydirs empties a directory of its objects and if that directory is part of a managed data structure (its objects are under revision control), then the populate operation removes the empty directory. If the empty directory is not part of a managed data structure, then the

```
                        operation does not remove the directory or its
                        subdirectories. (A directory is considered part
                        of the managed data structure if it has a
                        corresponding folder in the DesignSync vault
                        or if it contains a .SYNC client metadata
                        directory.)

                        Notes:
                          o When used with 'populate -force
                            -recursive', the -noemptydirs option removes
                            empty directories that have never been
                            managed.
                          o When used with the -mirror option, the
                            -noemptydirs option does not remove empty
                            directories (unless -force -recursive is
                            used) and does not populate directories that
                            are empty in the mirror.
                          o When the -noemptydirs option is used with
                            '-report verbose', the command might output
                            messages that additional directories are
                            being deleted. Those are directories created
                            by the populate, to mimic the directory
                            structure in the vault.  If no data is
                            fetched into those directories (because
                            no file versions match the selector),
                            then those empty directories are deleted.

                        If you do not specify -emptydirs or
                        -noemptydirs, the populate command follows
                        the DesignSync registry setting for "Populate
                        empty directories". By default, this setting
                        is not enabled; therefore, the populate
                        operation removes empty directories. To change
                        the default setting, your Synchronicity
                        administrator can use the SyncAdmin tool. For
                        information, see SyncAdmin help. You typically
                        want consistent behavior for all users, so
                        adding the setting to the site registry is
                        recommended.
```

-exclude (Module-based)

```
 -exclude <objects>  Specifies a comma-separated list of objects
                     (files, collections, folders, or module objects)
                     to be excluded from the operation. Wildcards are
                     allowed.

                     Note: Use the -filter option to filter module
                     objects.  You can use the -exclude option, but
                     the -filter option lets you include and exclude
                     module objects.  If you use both the -filter and
                     -exclude options, the strings specified using
                     -exclude take precedence.
```

If you exclude objects during a populate, a subsequent incremental populate will not necessarily process the folders of the previously excluded objects. DesignSync does not automatically perform a full populate in this case. To guarantee that previously excluded objects are fetched, specify the -full option for the subsequent populate operation.

The '-exclude' option is ignored if it is included in a 'populate -mirror' operation.

Do not specify paths in your arguments to -exclude. Before operating on each object (such as during a recursive populate), DesignSync compares the object's leaf name (with the path stripped off) to the exclude list to see if there is a match. Because the object's path is removed, the object will not match any object in the exclude list specified with a path. For example, if you specify '-exclude bin/*.exe', you will not successfully exclude bin/foo.exe or any other *.exe file. You need to instead specify '-exclude *.exe', or '-exclude foo.exe' if you want to exclude only 'foo.exe'. The result is that you cannot exclude a specific instance of an object -- you exclude all matching objects.

In addition to objects you specify using the -exclude option, the field, "These objects are always excluded", from the Tools->Options-> General->Exclude Lists dialog box in DesignSync lists objects that are always excluded from revision-control operations.

Note: Do not exclude members when you are fetching a module into the module cache; users cannot link to or copy from a filtered module in a module cache.

-exclude (File-based)

-exclude <objects> Specifies a comma-separated list of objects (files, collections, or folders) to be excluded from the operation. Wildcards are allowed.

If you exclude objects during a populate, a subsequent incremental populate will not necessarily process the folders of the previously excluded objects. DesignSync does not automatically perform a full populate in this case. To guarantee that previously excluded objects are fetched, specify the

```
                    -full option for the subsequent populate
                    operation.

                    The '-exclude' option is ignored if it is
                    included in a 'populate -mirror' operation.

                    Do not specify paths in your arguments to
                    -exclude. Before operating on each object (such
                    as during a recursive populate), DesignSync
                    compares the object's leaf name (with the path
                    stripped off) to the exclude list to see if there
                    is a match. Because the object's path is removed,
                    the object will not match any object in the
                    exclude list specified with a path. For example,
                    if you specify '-exclude bin/*.exe', you will not
                    successfully exclude bin/foo.exe or any other
                    *.exe file. You need to instead specify '-exclude
                    *.exe', or '-exclude foo.exe' if you want to
                    exclude only 'foo.exe'. The result is that you
                    cannot exclude a specific instance of an object
                    -- you exclude all matching objects.

                    In addition to objects you specify using the
                    -exclude option, the field, "These objects are
                    always excluded", from the Tools->Options->
                    General->Exclude Lists dialog box in
                    DesignSync lists objects that are always
                    excluded from revision-control operations.
```

-filter (Module-based)

```
 -filter <string>   Specify one or more extended glob-style
                    expressions to identify an exact subset of module
                    objects on which to operate. Use the -exclude
                    option to filter out DesignSync objects that are
                    not module objects.

                    The -filter option takes a list of expressions
                    separated by commas, for example:
                      -filter +top*/.../*.v,-.../a*

                    Prepend a '-' character to a glob-style expression
                    to identify objects to be excluded (the default).
                    Prepend a '+' character to a glob-style expression
                    to identify objects to be included.  Note that if
                    the list of expressions begins with an include
                    character ('+'), the filter excludes all objects
                    except those that match the include string.

                    Specify the paths in your glob-style expressions
                    relative to the current directory, because
                    DesignSync matches your expressions relative to
                    that directory. For submodules followed through
                    hrefs, DesignSync matches your expressions against
```

the objects' natural paths -- their full relative paths. For example, if a module, Chip, references a submodule, CPU, and CPU contains a file, '/libs/cpu/cdsinfo.tag', DesignSync matches against '/libs/cpu/cdsinfo.tag', rather than matching directly within the 'cpu' directory.

If your design contains symbolic links that are under revision control, DesignSync matches against the source path of the link rather than the dereferenced path. For example, if a symbolic link exists from 'tmp.txt' to 'tmp2.txt', DesignSync matches against 'tmp.txt'. Similarly for hierarchical operations, DesignSync matches against the unresolved path. If, for example, a symbolic link exists from dirA to dirB and dirB contains 'tmp.txt', DesignSync matches against 'dirA/tmp.txt'.

The extended glob-style expressions you use to filter the objects are standard glob-style expressions, but they are extended so that you can use the "..." syntax to indicate that the expression matches any number of directory levels. For example, the expression, "top/.../lib/*.v" matches *.v files in a directory path that begins with "top", followed by zero or more levels, with one of those levels containing a lib directory. The command traverses the directory structure. If a directory name matches an exclude clause of the filter, then the entire directory and all its contents are filtered (the command stops descending at that point), otherwise the command continues traversing the directory structure searching for matching objects.

If this is the first time a module is being populated, the filter becomes a persistent filter for the module, just as if you had invoked the setfilter command. For subsequent operations on the module, DesignSync applies persistent filters first, followed by those set using the -filter, -hreffilter, and -exclude options.

Note: If a populate specifies a -filter value to filter out objects that were previously populated, the populate is not considered complete. In this case, the workspace module does not match the module in the vault; thus, the module version is not updated. Also, a subsequent incremental populate will not necessarily process the folders of the previously excluded objects. DesignSync does not automatically perform a full populate in this case. To guarantee that previously excluded objects are fetched, specify the

```
-full option for the subsequent populate
operation.

Although the -filter option takes precedence over
persistent filters, it does not override the
exclude list set using SyncAdmin's
General=>Exclude Lists tab; the items in the
exclude list are combined with the filter
expression.  For example, an exclude list of
"*%,*.reg" combined with '-filter .../*.doc' is
equivalent to:
'-filter .../*.doc,.../*%,.../*.reg'.

Note: Do not filter a module that you are fetching
into the module cache; users cannot link to or
copy from a filtered module in a module cache.
```

-[no]force (Module-based)

```
-[no]force      Specifies whether to overwrite locally modified
               objects in order to match the workspace to the
               data being requested.  To do so, the populate
               operation deletes locally managed objects that
               are not part of the populate command line,
               deleting objects that have been filtered out.
               'populate -force' only removes managed data,
               not unmanaged data.  For module objects, the
               -force option removes objects from modules
               if they have been added by the add command, but
              have never been checked in.  Again, although
              DesignSync removes these objects from the module
               manifest, it does not remove the unmanaged data.
               Also, if you specify -force while populating
               a module that overlaps with another module,
               the -force option does not remove data from
               the other module.

               Use this option with caution, because you might
               not be able to retrieve lost changes.

               By default (-noforce):
               o Locally modified objects are not overwritten
                 by the populate operation. Specify -force if
                 you want to overwrite locally modified
                 objects. If the object is locked, the object
                 is unaffected by the populate operation
                 whether -force is specified or not.
               o Objects that are not part of the specified
                 module remain in your work area. If you
                 want to delete objects that are not part of
                 the configuration,   specify -force. Unmanaged
                 objects are never deleted.

               Using -force with -unifystate changes the state
```

of all objects including locally modified objects, in which case, local modifications are overwritten and objects are fetched according to the specified state or the default fetch state.

Using -force with -noemptydirs for populate removes all existing empty directories from the workspace unless the directories themselves are members of the module.

The -force option is mutually exclusive with both the -overlay and -noreplace options.

-[no]force (File-based)

-[no]force

Specifies whether to overwrite locally modified objects in order to match the workspace to the data being requested. To do so, the populate operation deletes locally managed objects that are not part of the populate command line, deleting objects that have been filtered out. 'populate -force' only removes managed data, not unmanaged data.

Use this option with caution, because you might not be able to retrieve lost changes.

By default (-noforce):

- Locally modified objects are not overwritten by the populate operation. Specify -force if you want to overwrite locally modified objects. If the object is locked, the object is unaffected by the populate operation whether -force is specified or not.
- Objects that are not part of the specified configuration remain in your work area. If you want to delete objects that are not part of the configuration, including retired objects, specify -force. Unmanaged objects are never deleted.

The behavior of 'populate -mirror' without -force is different from populate with other states (see the description of -mirror). Therefore, -force with -mirror has the additional effect of changing the state of existing objects in your work area, resulting in a hierarchy that exactly reflects the mirror directory.

Using -force with -unifystate changes the state of all objects including locally modified objects, in which case, local modifications are overwritten and objects are fetched according to

the specified state or the default fetch state.

Using -force with -noemptydirs for populate removes all existing empty directories from the workspace.

The -force option is mutually exclusive with both the -overlay and -noreplace options.

-from

-from <where>

Specifies whether the object is fetched from the vault ('-from vault') or from the cache or mirror ('-from local'). By default, DesignSync fetches from the cache or mirror ('-from local'), a performance optimization specific to the 'co -lock', 'co -get', 'populate -lock', and 'populate -get' commands. For details, see the Performance Optimization Overview in the DesignSync Data Manager Administrator's Guide. Note that this option is silently ignored when the optimization is not possible, including when the -keys option is specified.

The -from option can only be used with the -lock or -get fetch modes. It cannot be used with the -share, -mirror, -reference, or the -lock -reference combination fetch modes. If the -keys option is specified with the -from option, the -from option is silently ignored.

-full

-full

Performs a non-incremental populate by processing all objects and folders.

Note: DesignSync performs an incremental populate by default. It automatically reverts to a full populate when necessary. For more information, see the "Incremental Versus Full Populate" section in the description.

To change the default populate mode, your Synchronicity administrator can use the SyncAdmin tool.

Note: Do not use the -full option to change the states of objects in your work area (for example, changing from locked to unlocked objects or unlocked objects to links to

the cache). DesignSync changes the states of only those objects that need an update. Use the -unifystate option to change the state of objects in your work area.

-get (Module-based)

-get

Fetch unlocked copies.

You can change whether the local object is read-only (typical when using the locking model) or read/write (typical when using the merging model) by default by using the "Check out read only when not locking" option from the Tools->Options->General dialog box in the DesignSync graphical interface. Your project leader can also set this option site-wide using SyncAdmin.

This option is the default object-state option unless a default fetch preference has been defined. See the "fetch preference" help topic for more information.

Using -force with -noemptydirs for 'populate -get' removes all existing empty directories. Using -force with -emptydirs, however, creates empty directories for corresponding empty vault folders. Note that the populate command ignores the -noemptydirs option when operating on modules, because folders are members of their corresponding modules and therefore cannot be removed.

The -get option is mutually exclusive with the other fetch modes: -lock, -share, -mirror, and -reference.

Note: To replace mcache links with physical copies of module members, use the -mcachemode server option,

-get (File-based)

-get

Fetch unlocked copies.

You can change whether the local object is read-only (typical when using the locking model) or read/write (typical when using the merging model) by default by using the "Check out read only when not locking" option

from the Tools->Options->General dialog box in the DesignSync graphical interface. Your project leader can also set this option site-wide using SyncAdmin.

This option is the default object-state option unless a default fetch preference has been defined. See the "fetch preference" help topic for more information.

Using -force with -noemptydirs for 'populate -get' removes all existing empty directories. Using -force with -emptydirs, however, creates empty directories for corresponding empty vault folders.

The -get option is mutually exclusive with the other fetch modes: -lock, -share, -mirror, and -reference.

-hreffilter (Module-based)

-hreffilter <string>	Excludes href values during recursive operations on module hierarchies. Because hrefs link to submodules, you use -hreffilter to exclude particular submodules. Note that unlike the -filter option which lets you include and exclude items, the -hreffilter option only excludes hrefs and, thus, their corresponding submodules. Note: When populating a workspace with symbolic links to a module cache, the -hreffilter option does not apply and is silently ignored. Specify the -hreffilter string as a glob-style expression. The href filter can be specified either as a simple href filter or as a hierarchical href filter. A simple href filter is a simple leaf module name; you cannot specify a path. DesignSync matches the specified href filter against hrefs anywhere in the hierarchy. Thus, DesignSync excludes all hrefs of this leaf name; you cannot exclude a unique instance of the href. A hierarchical href filter specifies a path and a leaf submodule, for example JRE/BIN excludes the BIN submodule only if it is directly beneath JRE in the hierarchy. When creating a hierarchical href filter, you do not specify the top-level module of the

hierarchy. If you want to filter using the top-level module, you begin the hreffilter with /, for example, "/JRE," would filter any JRE href referenced by the top-level module.

Note: You can use wildcards with both types of hreffilter, however, if a wildcard is used as the lone character in hierarchical href, it only matches a single level, for example: "JRE/*/BIN" would match a hierarchy like "JRE/SUB/BIN" but would not match "JRE/BIN" or "JRE/SUB/SUB2/BIN".

You can prepend the '-' exclude character to your string, but it is not required. Because the -hreffilter option only supports excluding hrefs, a '+' character is interpreted as part of the glob expression.

If this is the first time a module is being populated, the filter becomes a persistent filter for the module, just as if you had invoked the setfilter command. For subsequent operations on the module, DesignSync applies persistent filters first, followed by those set using the -filter, -hreffilter, and -exclude options.

Note: Hierarchical hreffilters can only be specified during an initial populate. To add, change, or remove a hierarchical hreffilter after the initial populate, you must use the setfilter command.

Whereas the -filter option can prevent a populate from being complete, thus preventing the version from being updated, the -hreffilter option does not prevent the version from being updated. The -hreffilter option prevents particular submodules from being fetched, but the failure to fetch a submodule does not affect the updating to a new version.

Note: Do not filter a module that you are fetching into the module cache; users cannot link to or copy from a filtered module in a module cache.

-hrefmode (Module-based)

-hrefmode

For a recursive populate, determines whether to populate statically-specified submodules or dynamically-evaluated submodules.

Valid values are:
- dynamic - Expands hrefs at the time of the populate operation to identify the version

of the submodules to be populated.
- static - Populates with the submodules versions referenced by the hrefs when the module version was initially created.
- normal - Expands the hrefs at the time of the populate operation until it reaches a static selector. If the reference uses a static version, the hrefmode is set to 'static' for the next level of submodules to be populated; otherwise, the hrefmode remains 'normal' for the next level. (Default). This behavior can be changed using the "HrefModeChangeWithTopStaticSelector" registry key to determine how hrefs are followed.

Notes:
- If the -hrefmode option is used, it is stored for subsequent populates; You do not have to specify the href mode again unless a different mode is required.
- Use of the -hrefmode option is mutually exclusive with use of the -lock option.
- If an href is created with a mutable version tag, and that version tag has moved, you must use dynamic mode (-hrefmode dynamic) to populate your workspace with the new tagged version. If you want the workspace to continue to point to the original version, you should populate with normal or static mode.
- If you are fetching modules into the module cache, use the static mode (-herfmode static). You can only link to statically fetched module versions. See DesignSync Data Manager Administrator's Guide: "Setting up a Module Cache" for more information.

-incremental

-incremental

Performs a fast populate operation by updating only those folders whose corresponding vault folders contain modified objects.
Note: DesignSync performs an incremental populate by default. It automatically reverts to a full populate when necessary.
For more information, see the "Incremental Versus Full Populate" section in the description.

To change the default populate mode, your Synchronicity administrator can use the SyncAdmin tool.

Note: Do not use the -incremental option to change the states of objects in your work area (for example, changing from locked to unlocked objects or unlocked objects to links to the cache). DesignSync changes the states of updated objects only. For an incremental populate, DesignSync only processes folders that contain objects that need an update. State changes, therefore are not guaranteed. Use the -unifystate option to change the state of objects in your work area.

-keys (Module-based)

-keys <mode>

Controls processing of vault revision-control keywords in populated objects. Note that keyword expansion is not the same as keyword update. For example, the $Date$ keyword is updated only during checkin; its value is not updated during checkout or populate. The -keys option only works with the -get and -lock options. If you use the -share or -mirror option, keywords are automatically expanded in cached or mirrored objects, as if the '-keys kkv' option was used.

Available modes are:

kkv - (keep keywords and values) The local object contains both revision control keywords and their expanded values; for example, $Revision: 1.4 $.

kk - (keep keywords) The local object contains revision control keywords, but no values; for example, $Revision$. This option is useful if you want to ignore differences in keyword expansion, such as when comparing two different versions of an object.

kv - (keep values) The local object contains expanded keyword values, but not the keywords themselves; for example, 1.4. This option is not recommended if you plan to check in your local objects. If you edit and then check in the objects, future keyword substitution is impossible, because the value without the keyword is interpreted as regular text.

ko - (keep output) The local object contains the same keywords and values as were present at check in.

The -keys option can only be used with the -lock or -get fetch modes. It cannot be used with the -share, -mirror, -reference, or the -lock -reference combination fetch modes. If the -keys option is specified with the -from option, the -from option is silently ignored.

Note: When a module member is checked out with a lock, the locker keyword is not updated for the lock operation and remains null.

-keys (File-based)

-keys <mode>

Controls processing of vault revision-control keywords in populated objects. Note that keyword expansion is not the same as keyword update. For example, the $Date$ keyword is updated only during checkin; its value is not updated during checkout or populate. The -keys option only works with the -get and -lock options. If you use the -share or -mirror option, keywords are automatically expanded in cached or mirrored objects, as if the '-keys kkv' option was used.

Available modes are:

kkv - (keep keywords and values) The local object contains both revision control keywords and their expanded values; for example, $Revision: 1.4 $.

kk - (keep keywords) The local object contains revision control keywords, but no values; for example, $Revision$. This option is useful if you want to ignore differences in keyword expansion, such as when comparing two different versions of an object.

kv - (keep values) The local object contains expanded keyword values, but not the keywords themselves; for example, 1.4. This option is not recommended if you plan to check in your local objects. If you edit and then check in the objects, future keyword substitution is impossible, because the value without the keyword is interpreted as regular text.

ko - (keep output) The local object contains the same keywords and values as were present at check in.

The -keys option can only be used with the -lock or -get fetch modes. It cannot be used with the

-share, -mirror, -reference, or the -lock -reference combination fetch modes. If the -keys option is specified with the -from option, the -from option is silently ignored.

-lock (Module-based)

-lock

Lock the branch of the specified version for each module member object that is populated. Only the user who has the lock can check in a newer version of the object on that branch.

The -lock option does not lock not the module branch. In so doing, the -lock option makes the members writable in the workspace, and converts cached objects to full copies. To lock the module branch itself without making members writable, use the lock command.

Use the -lock option with the -reference option to populate with locked references. For more information, see the -lock -reference option. Locked references are useful if you intend to generate objects and want to lock them before regenerating, as opposed to editing the previous version. Upon generation of the objects, they automatically become locked copies rather than locked references. Obtaining locked references for generated objects is faster because DesignSync does not fetch the previously generated objects. If the objects exist already in the workspace, DesignSync deletes them. If the objects exist and are locally modified, the operation fails. If you intend to overwrite the modifications, use -force to create the locked references. If the default fetch state is 'reference' and you specify the -lock option without the -reference option, DesignSync leaves locked copies of the objects in your workspace; you must explicitly apply the -reference option with the -lock option if you want locked references in your workspace.

The -lock option is mutually exclusive with the fetch modes: -get, -share, and -mirror and mutually exclusive with -recursive. The -lock option can be used with the -merge option.

Notes:

- If you specify 'populate -lock', then by default the populate operation also uses the '-from local' option. The result is that the populate operation locks the object in the

vault and keeps local modifications in your workspace. See the -from option for information.

o When a module member is checked out with a lock, the locker keyword is not expanded with the locker name.

-lock (Legacy-based)

-lock

Lock the branch of the specified version for each object that is populated. Only the user who has the lock can check in a newer version of the object on that branch.

Use the -lock option with the -reference option to populate with locked references. For more information, see the -lock -reference option. Locked references are useful if you intend to generate objects and want to lock them before regenerating, as opposed to editing the previous version. Upon generation of the objects, they automatically become locked copies rather than locked references. Obtaining locked references for generated objects is faster because DesignSync does not fetch the previously generated objects. If the objects exist already in the workspace, DesignSync deletes them. If the objects exist and are locally modified, the operation fails. If you intend to overwrite the modifications, use -force to create the locked references. If the default fetch state is 'reference' and you specify the -lock option without the -reference option, DesignSync leaves locked copies of the objects in your workspace; you must explicitly apply the -reference option with the -lock option if you want locked references in your workspace.

The -lock option is mutually exclusive with the fetch modes: -get, -share, and -mirror, and with -merge option.

Notes:

o If you specify 'populate -lock', then by default the populate operation also uses the '-from local' option. The result is that the populate operation locks the object in the vault and keeps local modifications in your workspace. See the -from option for information.

o If you use 'populate -lock -recursive' to fetch or update a module configuration

hierarchy, populate locks only the objects associated with the upper-level module (the module configuration specified as the target of the command).

-lock (File-based)

-lock

Lock the branch of the specified version for each object that is populated. Only the user who has the lock can check in a newer version of the object on that branch.

Use the -lock option with the -reference option to populate with locked references. For more information, see the -lock -reference option. Locked references are useful if you intend to generate objects and want to lock them before regenerating, as opposed to editing the previous version. Upon generation of the objects, they automatically become locked copies rather than locked references. Obtaining locked references for generated objects is faster because DesignSync does not fetch the previously generated objects. If the objects exist already in the workspace, DesignSync deletes them. If the objects exist and are locally modified, the operation fails. If you intend to overwrite the modifications, use -force to create the locked references. If the default fetch state is 'reference' and you specify the -lock option without the -reference option, DesignSync leaves locked copies of the objects in your workspace; you must explicitly apply the -reference option with the -lock option if you want locked references in your workspace.

The -lock option is mutually exclusive with the fetch modes: -get, -share, and -mirror and with the -merge option.

Notes:

- If you specify 'populate -lock', then by default the populate operation also uses the '-from local' option. The result is that the populate operation locks the object in the vault and keeps local modifications in your workspace. See the -from option for information.

-lock -reference (Module-based)

-lock -reference

Use the -lock option with the -reference option to populate with locked references. Locked references are useful if you intend to generate objects and want to lock them before regenerating, as opposed to editing the previous version. Upon generation of the objects, they automatically become locked copies rather than locked references. Obtaining locked references for generated objects is faster because DesignSync does not fetch the previously generated objects. If the objects exist already in the workspace, DesignSync deletes them. If the objects exist and are locally modified, the operation fails. If you intend to overwrite the modifications, use -force to create the locked references. If the default fetch state is 'reference' and you specify the -lock option without the -reference option, DesignSync leaves locked copies of the objects in your workspace; you must explicitly apply the -reference option with the -lock option if you want locked references in your workspace.

The -lock -reference combination of option is mutually exclusive with the fetch modes: -get, -share, and -mirror, and with the -recursive option.

Note: You should not use the -reference option with Cadence data collection objects. When the -reference option is used on Cadence collections, DesignSync creates a reference in the metadata for the collection object but member files are not processed and are not included in the metadata.

-lock -reference (File-based)

-lock -reference

Use the -lock option with the -reference option to populate with locked references. Locked references are useful if you intend to generate objects and want to lock them before regenerating, as opposed to editing the previous version. Upon generation of the objects, they automatically become locked copies rather than locked references. Obtaining locked references for generated objects is faster because DesignSync does not fetch the previously generated objects. If the objects exist already in the workspace, DesignSync deletes them. If the objects exist and are locally modified, the operation fails. If you intend to overwrite the modifications, use -force to create the

locked references. If the default fetch state is 'reference' and you specify the -lock option without the -reference option, DesignSync leaves locked copies of the objects in your workspace; you must explicitly apply the -reference option with the -lock option if you want locked references in your workspace.

The -lock -reference combination of option is mutually exclusive with the fetch modes: -get, -share, and -mirror.

Note: You should not use the -reference option with Cadence data collection objects. When the -reference option is used on Cadence collections, DesignSync creates a reference in the metadata for the collection object but member files are not processed and are not included in the metadata.

-log

-log <filename>	Specify the name of the populate log file. If the filename doesn't exist, DesignSync creates it. If the file does exist, DesignSync appends the new information to the end of the log file. The filename can be specified with an absolute or relative path. If you specify a path for the log file, the directory you specify must already exist and you must have write permissions to the directory in order for the log to be placed into it, DesignSync does not create the path.

-mcachemode (Module-based)

-mcachemode <mcache_mode>	Specifies how the populate command fetches the module from the module cache. Note: The module cache should always be populated at the workspace root directory level. Available modes are: o link - For each module it finds in the module cache, the populate command sets up a symbolic link from your work area to the base directory of the module in the module cache. This is the default mode on UNIX platforms. Note:

```
    - This mode is supported on UNIX platforms
      only. If you specify link mode on a
      Windows platform, the populate operation
      fails.
   -  You cannot create mcache links to
      dynamically fetched modules since there is
      no auto-refresh of mcaches.
        the populate command.

o server - Causes the populate command to
  fetch modules as physical copies from the
  server, not the module cache. (Default for
  Windows.)

The -mcachemode option overrides the default
module cache mode registry setting. If
-mcachemode is not specified, the populate
command uses the mode specified in the registry
setting. If no registry setting is specified,
the command uses link mode on Unix platforms and
server mode on Windows platforms.

Notes on mcaches:
o If you run a populate with the -norecursive
  option, the module must have been fetched into
  the mcache in -norecursive mode as well, or
  the command will not create links to or copies
  from the module cache.

o If the populate command is run using a filter,
  no mcache link to or copies are made.
  Therefore a filtered module can never be used
  in an mcache even if populate is run in a
  workspace that uses the same filter.

o The mcache administrator can fetch modules
  into a module cache to link to or copy the
  contents of the module.

o You cannot create mcache links to mcache
  directories containing members of more than
  one module version.

If a request to link to the module cache is
disallowed, DesignSync fetches the module from
the server instead.

For more information using populate with a
module cache, see 'Module Caches' in the
description section of the populate command.
```

-mcachemode (Legacy-based)

```
-mcachemode          Specifies how the populate command fetches
```

<mcache_mode>

the legacy module from the module cache.

Note: The module cache should always be populated at the workspace root directory level.

Available modes are:

- link - For each module it finds in the module cache, the populate command sets up a symbolic link from your work area to the base directory of the module in the module cache. This is the default mode on UNIX platforms.

  Note:
  - This mode is supported on UNIX platforms only. If you specify link mode on a Windows platform, the populate operation fails.
  - You cannot create mcache links to dynamically fetched modules since there is no auto-refresh of mcaches.

- copy - For each module it finds in the module cache, the populate command copies the module to your work area. (Default on Windows platforms)

- server - Causes the populate command to fetch modules as physical copies from the server, not the module cache.

The -mcachemode option overrides the default module cache mode registry setting. If -mcachemode is not specified, the populate command uses the mode specified in the registry setting. If no registry setting is specified, the command uses link mode on Unix platforms and copy mode on Windows platforms.

Notes on module mcaches:

- The mcache administrator can fetch legacy modules into a legacy module cache to link to or copy the contents of the module.
- Legacy modules can be fetched into either a module cache or a legacy module cache by the mcache administrator, but they cannot be linked to or copied from.

If a request to link to or copy from the module cache is disallowed, DesignSync fetches the module from the server instead.

-mcachepaths (Module / Legacy-based)

-mcachepaths

Identifies one or more module caches to be searched for modules.

Path names can be absolute or relative. You can specify multiple paths in a list of space-separated path names. To specify multiple paths, surround the path list with double quotation marks ("") and separate path names with a space. For example:"/dir/cacheA /dir2/cacheB".

The path list can contain both module and legacy module mcache paths. For a module cache the path to the root directory of the module cache must be supplied.

This option overrides the default module cache paths registry setting. If -mcachepaths is not specified, the command uses the list of paths specified in the registry setting. If no registry setting is specified, the populate command fetches modules from the server.

Note:
o To specify a path that includes spaces:
  - In stcl or stclc, surround the path containing the spaces with curly braces. For example:
    "/dir1/cache {/dir2/path name}"
  - In dss or dssc, use backslashes (\) to 'escape' the spaces. For example:
    "/dir1/cache /dir2/path\ with\ spaces"
o The populate command searches the mcache in the order specified with the -mcachepaths option or in the default module cache paths registry setting if this option is absent.

-[no]merge (Module-based)

-[no]merge

Indicates whether to populate with the Latest versions from the branch specified by the persistent selector list and merge them with the current, locally modified versions. The default value is -nomerge.

If you are not doing an overlay merge (see -overlay) and the current version is not locally modified, the -merge defaults to a -get and fetches the new version without merging. By definition, a merge expects a locally modified object, so the -force option is not required.

The -merge option supports the merging work model (as opposed to the locking work model) where multiple team members can check out and edit the Latest version concurrently. The first team member to check in creates the next version. Other team members must merge the new Latest version into their local copy before they can check in their changes.

If there are no merge conflicts, the merge succeeds, leaving the merged files in your work area. If there are conflicts, DesignSync issues a warning, and you must edit the merged file to resolve the conflicts before DesignSync allows you to check in the merged version. Conflicts are shown as follows:

```
<<<<<<< local
Lines from locally modified version
=======
Lines from selected server version
>>>>>>> versionID
```

DesignSync considers the conflicts resolved when the file no longer contains any of the conflict delimiters (exactly 7 less-than, greater-than, or equal signs starting in column 1). The status of an object, as displayed by ls or from the List View in the DesignSync graphical interface, indicates if conflicts exist. The url inconflict command also determines whether a file has conflicts.

Most merges are between two versions on the same branch (the current branch and the branch specified by the persistent selector list are typically the same). However, a merge can also be performed across branches by setting the persistent selector list to a different branch. Following the merge, you are on the branch associated with the version specified by the persistent selector list (a 'merge to' operation). If you want to stay on the current branch instead, use the -overlay option. Overlay ('merge from') merges are more common when merging branches. See the -overlay option for details.

Note:
o When merging modules across branches, you should use -merge -overlay. For details about merging modules across branches, see the "Merging Across Branches section."
o The -merge option implies -get, but you can

also explicitly specify -get. For general DesignSync objects, the -merge option is mutually exclusive with all other state options (-lock, -share, -mirror, -reference, and -lock -reference).
You can use -lock with -merge for modules and their members.

o The -merge and -version options are mutually exclusive unless you specify '-version Latest'.

-merge (File-based)

-[no]merge

Indicates whether to populate with the Latest versions from the branch specified by the persistent selector list and merge them with the current, locally modified versions. The default value is -nomerge.

If you are not doing an overlay merge (see -overlay) and the current version is not locally modified, the -merge defaults to a -get and fetches the new version without merging. By definition, a merge expects a locally modified object, so the -force option is not required.

The -merge option supports the merging work model (as opposed to the locking work model) where multiple team members can check out and edit the Latest version concurrently. The first team member to check in creates the next version. Other team members must merge the new Latest version into their local copy before they can check in their changes.

If there are no merge conflicts, the merge succeeds, leaving the merged files in your work area. If there are conflicts, DesignSync issues a warning, and you must edit the merged file to resolve the conflicts before DesignSync allows you to check in the merged version. Conflicts are shown as follows:

```
<<<<<<< local
Lines from locally modified version
=======
Lines from selected server version
>>>>>>> versionID
```

DesignSync considers the conflicts resolved when the file no longer contains any of the conflict delimiters (exactly 7 less-than,

greater-than, or equal signs starting in column 1). The status of an object, as displayed by ls or from the List View in the DesignSync graphical interface, indicates if conflicts exist. The url inconflict command also determines whether a file has conflicts.

Most merges are between two versions on the same branch (the current branch and the branch specified by the persistent selector list are typically the same). However, a merge can also be performed across branches by setting the persistent selector list to a different branch. Following the merge, you are on the branch associated with the version specified by the persistent selector list (a 'merge to' operation). If you want to stay on the current branch instead, use the -overlay option. Overlay ('merge from') merges are more common when merging branches. See the -overlay option for details.

Note:
- The -merge option implies -get, but you can also explicitly specify -get. For general DesignSync objects, the -merge option is mutually exclusive with all other state options (-lock, -share, -mirror, -reference, and -lock -reference).
  You can use -lock with -merge for modules and their members.
- The -merge and -version options are mutually exclusive unless you specify '-version Latest'.

-mirror (File-based)

-mirror

Create symbolic links from the work area to objects in the mirror directory. This option requires that you have associated a mirror directory with your work area (see the 'setmirror' command).

For performance reasons, links are created only when objects do not exist in your work area. To update mirror links for existing objects, use -unifystate with the -mirror option. For example:

```
populate -recursive -full -unifystate -mirror
```

The -unifystate option does not affect locally modified objects or objects that are not part of the configuration. Use -force with

-unifystate to update the links, replacing locally modified objects and removing objects that are not part of the current configuration.

When used with the -mirror option, the -noemptydirs option does not populate directories that are empty on the mirror. Using the -force option with the -noemptydirs option removes all empty directories from the workspace. Using -force with -emptydirs for 'populate -mirror', however, populates empty directories that exist in the mirror.

The -mirror option is mutually exclusive with the other fetch modes: -lock, -get, -share, and -reference. The -mirror option is also mutually exclusive with the -keys and -from options. The -mirror option cannot take an exclude filter. If the -exclude option is specified with the -mirror fetch mode, the populate silently ignores the -exclude option.

Note:
- This option is not supported on Windows platforms.
- The -exclude option is ignored if it is included in a 'populate -mirror' operation.
- If you specify -mirror, an incremental populate does not necessarily fetch new objects checked in, nor remove links to objects deleted by team members until after the mirror is updated.
- When populating a custom generic collection from a mirror, always use 'populate -mirror' from the folder containing the collection object or from a folder above the folder containing the object.

-modulecontext (Module-based)

-modulecontext — Identifies the module to be populated. Use the -modulecontext option if your workspace has overlapping modules, so that you can indicate which module to populate.

You can use the -modulecontext option when specifying a folder to populate. In this case, the populate operation filters the folder, populating only those objects that belong to the module specified with the -modulecontext option. Use -modulecontext in a recursive populate to fetch members of the specified module throughout a hierarchy.

You can also use -modulecontext option to

identify which module to fetch items from when requesting an object that is not currently in the module.

Specify an existing workspace module using the module name (for example, Chip) or a module instance name (for example, Chip%0). You also can specify -modulecontext as a server module URL (sync://server1:2647/Modules/Chip).

Notes:
o You cannot use a -modulecontext option to operate on objects from more than one module; the -modulecontext option takes only one argument, and you can use the -modulecontext option only once on a command line.

o If you have overlapping modules, you must specify -modulecontext when populating a module that contains files not present in your workspace.

-[no]new (Module-based)

-[no]new — Specifies whether to fetch module objects that are not yet in the workspace.

Apply the -new (default) to fetch all specified module objects (except those filtered out by options such as -filter and -exclude). Specify -nonew option to update only those objects already in the workspace.

Using -new is another form of filtering. It can cause the subsequent populate to be a full rather than an incremental populate.

Note: This option is supported for module objects only.

-overlay

-overlay <selectors> — Replace your local copy of the module or DesignSync non-module object with the versions specified by the selector list (typically a branch tag). The current-version status, as stored in local metadata, is unchanged. For example, if you have version 1.5 (the Latest version) of the module or DesignSync object and you overlay version 1.3, your current version is still 1.5. You could then check in this overlaid version. This operation is equivalent

to checking out version 1.3, then using 'ci -skip' to check in that version.

The behavior of the overlay operation depends on the presence of a local version and the version you want to overlay:

- If both the local version and the overlay version exist, the local version is replaced by the overlay version.
- If there is no local version but an overlay version exists, DesignSync creates a local copy of the overlay version.
- If a local version exists but there is no overlay version, the local version is unaffected by the operation.
- If the overlay version was renamed or removed, the local object is not changed, but metadata is added to it, indicating the change. This information can be viewed using the ls command with the -merged option.

Typically, you use -overlay with -merge to merge the two versions instead of overlaying one version onto another. The combination of -overlay and -merge lets you merge from one branch to another, the recommended method for merging across branches. Following the overlay merge, you are working on the same branch as before the operation.

You specify the version you want to overlay as an argument to the -overlay option. The -overlay and -version options are mutually exclusive. The -version option always updates the 'current version' information in your work area, which is not correct for an overlay operation.

- To use -overlay to specify a branch, specify both the branch and version as follows: '<branchtag>:<versiontag>', for example, 'Rel2:Latest'. You can also use the shortcut, '<branchtag>:', for example "Rel2:". If you don't explicitly specify the branch selector in this way, DesignSync does not resolve the selector as a branch selector.

When doing an overlay (with or without -merge), a number of combinations for the state of a module or DesignSync object on the two branches must be considered. For more information, see the "Merging Across Branches" section above. Hierarchical references in modules are not updated during an overlay.

Notes:

```
                    o The -overlay option implies -get, but
                      you can also explicitly specify -get.
                    o The -overlay option is mutually exclusive
                      with the other state options (-mirror,
                      -share, -lock, -reference) and -version.
```

-path (Module-based)

```
 -path <path>       Specify the name of an alternate local folder
                    to populate instead of the current folder. The
                    populate command uses the vault and persistent
                    selector list associated with the specified
                    folder.
                    Note: Using -path is equivalent to changing
                          folders, executing the populate
                          command, then changing back to the
                          original folder.

                    If you specify a folder using the -path option
                    but the folder does not exist, DesignSync
                    - verifies that a corresponding vault exists
                    - creates the folder
                    - populates the specified folder, creating
                    any interim folders necessary to replicate
                    the vault hierarchy locally.

                    If you specify the -target option but the
                    folder does not exist, DesignSync creates
                    the folder.
                    Generally, however, if the vault does not
                    exist, DesignSync does not create the
                    folder and leaves the workspace unchanged.

                    Tip: When populating a workspace with links to a
                    module cache, use -path to create the directory,
                    rather than specifying an existing directory.

                    The -path option used to be the -dir option.
                    The -dir option is still provided for backwards
                    compatibility, but is not documented
                    separately.
```

-path (Legacy-based)

```
 -path <path>       Specify the name of an alternate local folder
                    to populate instead of the current folder. The
                    populate command uses the vault and persistent
                    selector list associated with the specified
                    folder.
                    Note: Using -path is equivalent to changing
                          folders, executing the populate
                          command, then changing back to the
```

```
      original folder.

If you specify a folder using the -path option
but the folder does not exist, DesignSync
- verifies that a corresponding vault exists
- creates the folder
- populates the specified folder, creating
any interim folders necessary to replicate
the vault hierarchy locally.

If you specify the -target option but the
folder does not exist, DesignSync creates
the folder.
Generally, however, if the vault does not
exist, DesignSync does not create the
folder and leaves the workspace unchanged.

Note: If the folder specified by -path does not
exist, but corresponds to a vault with
unpopulated legacy modules or DesignSync
REFERENCES, DesignSync has no way to resolve
these mappings. In this case, populate does not
create the specified folder, leaving the
workspace unchanged.

The -path option used to be the -dir option.
The -dir option is still provided for backwards
compatibility, but is not documented
separately.
```

-path (File-based)

```
-path <path>        Specify the name of an alternate local folder
                    to populate instead of the current folder. The
                    populate command uses the vault and persistent
                    selector list associated with the specified
                    folder.
                    Note: Using -path is equivalent to changing
                          folders, executing the populate
                          command, then changing back to the
                          original folder.

                    If you specify a folder using the -path option
                    but the folder does not exist, DesignSync
                    - verifies that a corresponding vault exists
                    - creates the folder
                    - populates the specified folder, creating
                    any interim folders necessary to replicate
                    the vault hierarchy locally.

                    If you specify the -target option but the
                    folder does not exist, DesignSync creates
                    the folder.
                    Generally, however, if the vault does not
```

exist, DesignSync does not create the folder and leaves the workspace unchanged.

Note: If the folder specified by -path does not exist, but corresponds to a vault with unpopulated DesignSync REFERENCES, DesignSync has no way to resolve these mappings. In this case, populate does not create the specified folder, leaving the workspace unchanged.

The -path option used to be the -dir option. The -dir option is still provided for backwards compatibility, but is not documented separately.

-[no]recursive (Module-based)

-[no]recursive

Specifies whether to perform this operation on the specified folder or module only (default), or to traverse its subfolders or submodules.

If you invoke 'populate -recursive' and specify a folder, populate operates on the folder in a folder-centric fashion, fetching the objects in the folder and its subfolders.
If the folders or subfolders contain modules or module members, populate fetches the objects, but does not follow hierarchical references (hrefs). To filter the set of objects on which to operate, use the -filter or -exclude options.

If you invoke 'populate -recursive' and specify a module, populate operates on the specified module in a module-centric fashion, fetching all of the objects in the module and following its hierarchical references (hrefs) to fetch its referenced submodules. To filter the objects on which to operate, use the -filter or -hreffilter options.

Note: Because of the way module merge handles hierarchical reference, you cannot specify -recursive when doing a cross branch merge on a module, (pop -merge -overlay).

If you invoke 'populate -recursive' on a subfolder of a module and provide a -modulecontext, populate recurses within the specified folder, fetching any object which is a member of the named module or one of its referenced submodules.
Note: For modules, you cannot use the -recursive option with the -lock option.

Note: The populate operation might skip

subfolders and individual managed objects
if their persistent selector lists differ
from the top-level folder being populated;
see the Description section for details.

If you specify -norecursive when operating on
a folder, DesignSync operates only on objects in
the specified folder. In this case, populate
does not traverse the vault folder hierarchy.
Likewise, if you specify -norecursive when
operating on a module, DesignSync operates
only on the module objects and does not follow
hrefs.

-[no]recursive (Legacy-based)

-[no]recursive

Specifies whether to perform this operation on
the specified folder only (default), or to
traverse its subfolders or hierarchical
references.

If you invoke 'populate -recursive' and specify a
folder, populate operates on the folder in a
folder-centric fashion, fetching the objects in
the folder and its subfolders. It does not follow
the hierarchical references (hrefs). To filter
the set of objects on which to operate, use the
-exclude option.

Note: The populate operation might skip
subfolders and individual managed objects
if their persistent selector lists differ
from the top-level folder being populated;
see the Description section for details.

If you specify -norecursive when operating on
a folder, DesignSync operates only on objects in
the specified folder. In this case, populate
does not traverse the vault folder hierarchy.

If you perform a -norecursive populate, then for
the subsequent populate DesignSync performs a
full populate even if the -full option is not
specified.

Notes:
- DesignSync cannot perform an incremental
  populate following a nonrecursive populate,
  because it cannot ensure that the objects in
  the work area subfolders are up-to-date.

- The -nomodulerecursive option is no longer
  required. If you apply the -nomodulerecursive
  option to legacy modules, populate recurses

within the legacy module's folders. It does not traverse REFERENCEs or hrefs of legacy modules.

-[no]recursive (File-based)

-[no]recursive

Specifies whether to perform this operation on the specified folder (default), or to traverse its subfolders.

If you invoke 'populate -recursive' and specify a folder, populate operates on the folder in a folder-centric fashion, fetching the objects in the folder and its subfolders. To filter the set of objects on which to operate, use the -exclude option.

Note: The populate operation might skip subfolders and individual managed objects if their persistent selector lists differ from the top-level folder being populated; see the Description section for details.

If you specify -norecursive when operating on a folder, DesignSync operates only on objects in the specified folder. In this case, populate does not traverse the vault folder hierarchy.

If you perform a -norecursive populate, then for the subsequent populate DesignSync performs a full populate even if the -full option is not specified.

Note: DesignSync cannot perform an incremental populate following a nonrecursive populate, because it cannot ensure that the objects in the work area subfolders are up-to-date.

-reference

-reference

Populate with DesignSync references to objects in the vault. A reference does not have a corresponding file on the file system but does have local metadata that makes the reference visible to Synchronicity programs. Populate with references when you want your work area to reflect the contents of the vault but you do not need physical copies. Use the -reference option with the -lock option to populate with locked references. Locked references are useful if you intend to generate objects and want to lock them before regenerating,

as opposed to editing the previous versions.

Note: You should not use the -reference option with Cadence data collection objects. When the -reference option is used on Cadence collections, DesignSync creates a reference in the metadata for the collection object but member files are not processed and are not included in the metadata.

-[no]replace (Module-based)

-[no]replace

This option determines how to handle locally modified objects when synchronizing your work area.

The -replace option specifies that the populate operation updates locally unmodified workspace objects. This option leaves intact all managed objects that are not members of the module (if applicable) and all unmanaged objects. If an object has been removed from the version being fetched as a result of a remove operation or retired on the server, -replace removes the member from the workspace if it has not been locally modified. (Default)

The -noreplace option specifies that the populate operation updates managed objects that have not been locally modified. The -noreplace option leaves intact all unmanaged objects. If an object has been removed from the version being fetched as a result of a remove, mvmember, rmhref or any other similar operation, -noreplace does not remove the corresponding file in the workspace.

During a recursive populate, -noreplace leaves intact managed objects belonging to a referenced submodule even when the href has been removed. If the href has been changed to reference a different submodule, -noreplace:

- Leaves intact managed objects that belong to the previous submodule but not to the new submodule
- Replaces managed members that belong to both modules with the version belonging to the new module

Notes:

- See "Forcing, Replacing, and Non-Replacing Modes" above to see how the -force option interacts with the -[no]replace option.

- If you use populate -version to populate a directory containing a module, DesignSync uses the -noreplace option unless -replace is explicitly specified.
- If you apply the -filter or -hreffilter options, populate applies the -[no]replace option on the filtered data.
- With a recursive operation, populate applies -replace and -noreplace behaviors to the top-level module and then to each referenced submodule.

-[no]replace (File-based)

-[no]replace

This option determines how to handle locally modified objects when synchronizing your work area.

The -replace option specifies that the populate operation updates locally unmodified workspace objects. This option leaves intact all managed objects and all unmanaged objects. If an object has been removed from the vault being fetched as a result of a retire, rmvault, or any other similar operation, -replace removes the file from the workspace if it has not been locally modified.

The -noreplace option specifies that the populate operation updates managed objects that have not been locally modified. The -noreplace option leaves intact all unmanaged objects. If an object has been removed from the vault being fetched as a result of a retire, rmvault, or any other similar operation, -noreplace does not remove the corresponding file in the workspace. (Default)

Notes:

- See "Forcing, Replacing, and Non-Replacing Modes" above to see how the -force option interacts with the -[no]replace option.
- If you use populate -version to populate a directory containing a module, DesignSync uses the -noreplace option unless -replace is explicitly specified.
- If you apply the -filter or -hreffilter options, populate applies the -[no]replace option on the filtered data.
- With a recursive operation, populate applies -replace and -noreplace behaviors to the top-level module and then to each referenced submodule.

-report

-report error| brief|normal| verbose	Specifies the amount and type of information displayed by the command. The information each option returns is discussed in detail in the "Understanding the Output" section above. error - lists failures, warnings, and success failure count. brief - lists failures, warnings, module create/remove messages, some informational messages, and success/failure count. normal - includes all information from brief, and lists all the updated objects, and messages about objects excluded by filters from the operation. (Default) verbose - provides full status for each object processed, even if the object is not updated by the operation.

-[no]retain

-[no]retain	Indicates whether to retain the 'last modified' timestamp of the fetched objects as recorded when each object was checked into the vault. If the workspace is set to use a mirror, or the populate is run using -share, this will also apply to the object placed in the mirror or LAN cache if the object doesn't already exist in the mirror or cache. The links in your work area to the cache or mirror have timestamps of when the links were created. If you specify the -reference option, no object is created in your work area, so there is no timestamp information at all. If an object is checked into the vault and the setting of the -retain option is the only difference between the version in the vault and your local copy, DesignSync does not include the object in populate operations. If you do not specify '-retain' or -noretain', the populate command follows the DesignSync registry setting for Retain last-modification timestamps. By default, this setting is not enabled; therefore, the timestamp of the local object is the time of the populate operation. To change the default setting, your

Synchronicity administrator can use the SyncAdmin tool. For information, see SyncAdmin Help.

The mirror system, by default,fetches objects into the mirror with the -retain option. The mirror administrator, however, can define mirrors to use the -noretain option. The default setting should agree with the Retain last-modification timestamp registry setting to maintain consistency. See the "Mirror Administration Server Registry Settings" topic for setting of the co or populate options for mirrors.

Note: When fetching from the cache or mirror (by specifying the '-from local' option), the last modified timestamp comes from the file in the cache or mirror, not from the version that was checked into the vault. If the file was fetched into the cache or mirror with the -retain option, these two timestamps are the same. But if the file was fetched into the cache or mirror with the -noretain option and then fetched into the workspace with both the '-from local' and '-retain' options, the 'last modified' timestamp used is the time the object was fetched into the cache or mirror.

-savelocal

-savelocal <value>

This option affects collections that have local versions.

When it fetches an object, the populate operation first removes from your workspace any local version that is unmodified. (To remove a local version containing modified data, specify 'pop -force'.) Then the populate operation fetches the object you are checking out (with the local version number it had at the time of checkin).

The -savelocal option specifies the action that the populate operation takes with modified local versions in your workspace (other than the current, or highest numbered, local version). (DesignSync considers a local version to be modified if it contains modified members or if it is not the local version originally fetched from the vault when the collection object was checked out or populated to your workspace.)

Specify the -savelocal option with one of the following values:

save - If your workspace contains a local version other than the local version being fetched, the populate operation saves the local version for later retrieval. See the 'localversion restore' command for information on retrieving local versions that were saved.

fail - If your workspace contains an object with a local version number equal to or higher than the local version being fetched, the populate operation fails. This is the default action.

Note: If your workspace contains an object with local version numbers lower than the local version being fetched and if these local versions are not in the DesignSync vault, the populate operation saves them. This behavior occurs even when you specify '-savelocal fail'

delete - If your workspace contains a local version other than the local version being fetched, the populate operation deletes the local version from your workspace.

If you do not specify the -savelocal option, the populate operation follows the DesignSync registry setting for SaveLocal. By default, this setting is "Fail if local versions exist" ('-savelocal fail'). To change the default setting, a Synchronicity administrator can use the Command Defaults options pane of the SyncAdmin tool. For information, see SyncAdmin Help.

Note:
- You may need to use the -force option with the -savelocal option to allow the object being fetched to overwrite a locally modified copy of the object. For an example scenario, see EXAMPLES.
- The -savelocal option affects only objects of a collection defined by the Custom Type Package (CTP). This option does not affect objects that are not part of a collection or collections that do not have local versions.

-share

-share

Fetch shared copies. Shared objects are stored in the file cache directory and links to the

cached objects are created in the work area.

Notes:
This option is not supported on Windows platforms.

The -share option is mutually exclusive with the other fetch modes: -lock, -get, -mirror, and -reference. The -share option is also mutually exclusive with the -keys and -from options.

-target (Legacy-based)

-target <server_module_url>	Specifies a legacy module configuration to fetch to your work area. Note: This option applies only to legacy modules. Also, this option is no longer required and will be removed in a future release; you can specify the module as a command argument. See ARGUMENTS above to specify the module as an argument. To specify a module using the -target option, use the syntax: sync[s]://<host>[:<port>]/<vaultPath> where <host> is the SyncServer on which the module resides, <port> is the SyncServer port number, and <vaultPath> is the vault folder in which the module's data resides. To specify a module configuration other than the default configuration, use the syntax: sync[s]://<host>[:<port>]/<vaultPath>@<config> where <host> is the SyncServer on which the module resides, <port> is the SyncServer port number, <vaultPath> is the vault folder in which the module's data resides, and <config> is the specific configuration of the module. If you specify this option, the populate command sets the vault and selector. If you specify the 'populate -target' with the -path option and the specified directory does not exist, the populate command creates the directory in your work area and sets the selector for fetching the configuration specified with '-target'. Note: To fetch an entire legacy module hierarchy, use the -recursive option with 'populate -target'. The 'populate -target' command checks whether the target is an ordinary DesignSync vault or a

module with no hrefs. In the cases where it is either a DesignSync Vault or a module with no hrefs and the registry setting indicates that the module with no hrefs should be treated like a DesignSync vault, it performs a setvault operation with the value specified to target and then performs an ordinary populate on the directory. Effectively, this is equivalent to performing a 'setvault' and populate (without -target). The setvault is done recursively if the -recursive option was specified with populate.

-trigarg

-trigarg <arg>

Specifies an argument to be passed from the command line to the triggers set on the populate operation. If the argument contains whitespace, enclose the argument within double quotation marks ("") if using the dss command shell or braces ({}) if using the stcl command shell.

-[no]unifystate

-[no]unifystate

Indicates whether to set the state of all objects processed, even up-to-date objects, to the specified state (-get, -lock, -share, -mirror, or -reference) or to the default fetch state if no state option is specified. See the "fetch preference" help topic for more information.

By default, populate changes the state of only those objects that are not up-to-date (-nounifystate). If the -unifystate option is specified, DesignSync changes the state of the up-to-date objects, as well, and thus performs a full populate.

The -unifystate option does not change the state of locally modified objects; use -force with -unifystate to force a state change, thus overwriting local modifications. The -unifystate option does not change the state of objects not in the configuration; use -force with -unifystate to remove objects not in the configuration.
The -unifystate option does not cancel locks; you can check in the locked files, use the 'cancel' command to cancel locks you have acquired, or use the 'unlock' command to cancel

team members' locks.

Note: The -unifystate option is ignored when you lock design objects. If you populate with locked copies or locked references, DesignSync leaves all processed objects in the requested state.

-version (Module-based)

-version <selector> Specifies the versions of the objects to populate. The selector list you specify (typically a version or branch tag) overrides the persistent selector lists of the objects you are populating. If you populate the top-level module in a hierarchy with the -version tag, you replace the persistent selector of the workspace with the version specified by this option. If you specify the -recursive option, the specified selector list is used to populate all subfolders during populates.

If you specify a date selector (Latest or Date(<date>)), DesignSync augments the selector with the persistent selector list to determine the versions to populate. For example, if the persistent selector list is 'Gold:,Trunk', and you specify 'populate -version Latest', then the selector list used for the populate operation is 'Gold:Latest,Trunk:Latest'.

For details on selectors and selector lists see the topic describing selectors.

Note:
- Using the -version option with the populate command changes the workspace selector if the populate was performed on a top-level module instance. If you are working in a module hierachy, you should use the swap replace command to change the sub-module version populated. If you populate individual module members or folders, the persistent selector is not updated.
- If you use -version to populate a module member, populate fetches the version that is appropriate to the module version as identified by the version value.
- If you use the -version option with the -incremental option, and the selector you specify does not exactly match the workspace selector, the incremental populate does not occur. DesignSync performs a full populate

```
                    instead. See "Incremental Versus Full
                    Populate" in the description section for more
                    information.
                  o When using -version to specify a branch,
                    specify both the branch and version as
                    follows: <branchtag>:<versiontag>, for
                    example, Rel2:Latest. You can also use the
                    shortcut, <branchtag>:, for example Rel2:.
                    If you do not explicitly specify the branch
                    selector in this way, DesignSync does not
                    resolve the selector as a branch selector.
                  o When you specify a version-extended name that
                    reflects the object's version, for example,
                    "file.txt;1.3", populate ignores the
                    -version option.
                  o Specify '-version <branchtag>:Latest' only if
                    necessary. In some cases, DesignSync augments
                    the selector to be <branchtag>:Latest.
                    When you append ':Latest', it may not match
                    the work area selector. This mismatch
                    invalidates your next incremental populate
                    resulting in a slower, full populate.
                  o The -version option is mutually exclusive
                    with -merge unless you specify '-version
                    Latest', the default.
                  o The -version and -overlay options are
                    mutually exclusive.
```

-version (File / Legacy-based)

```
 -version <selector>  Specifies the versions of the objects to
                      populate. The selector list you specify
                      (typically a version or branch tag) overrides
                      the persistent selector lists of the objects
                      you are populating.  If you specify the
                      -recursive option, the specified selector
                      list is used to populate all subfolders during
                      populate.
                      You can also specify a ProjectSync configuration;
                      see "Interaction with Legacy Modules" in the
                      Description section.

                      If you specify a date selector (Latest or
                      Date(<date>)), DesignSync augments the
                      selector with the persistent selector list
                      to determine the versions to populate. For
                      example, if the persistent selector list
                      is 'Gold:,Trunk', and you specify
                      'populate -version Latest', then the selector
                      list used for the populate operation is
                      'Gold:Latest,Trunk:Latest'.

                      For details on selectors and selector lists
                      see the topic describing selectors.
```

Note:
- Using the -version option with the populate command does not change the workspace selector, even during the initial populate of an object. To set the workspace selector as part of the populate command, specify the selector explicitly, using the <object>;<selector> syntax.
- If you use the -version option with the -incremental option, and the selector you specify does not exactly match the workspace selector, the incremental populate does not occur. DesignSync performs a full populate instead. See "Incremental Versus Full Populate" in the description section for more information.
- When using -version to specify a branch, specify both the branch and version as follows: <branchtag>:<versiontag>, for example, Rel2:Latest. You can also use the shortcut, <branchtag>:, for example Rel2:. If you do not explicitly specify the branch selector in this way, DesignSync does not resolve the selector as a branch selector.
- When you specify a version-extended name that reflects the object's version, for example, "file.txt;1.3", populate ignores the -version option.
- Specify '-version <branchtag>:Latest' only if necessary. In some cases, DesignSync augments the selector to be <branchtag>:Latest. When you append ':Latest', it may not match the work area selector. This mismatch invalidates your next incremental populate resulting in a slower, full populate.
- The -version option is mutually exclusive with -merge unless you specify '-version Latest', the default.
- The -version and -overlay options are mutually exclusive.
- When you use populate with the -version option to fetch a directory containing legacy modules, by default DesignSync uses the -noreplace

-view (Module-based)

`-view view1 [,view2[,view...]]`

Module view name or comma-delimited list of module view names, applied to a module or module hierarchy when it is fetched.

Note: This option is only valid for server module objects. If it is used with an argument

type other than a server module url, the option is silently ignored.

There is no default value for this option. You cannot set a default value in the command defaults system.

On an initial populate, the module view name or names list provided is propagated through the hierarchy and applied to all fetched modules. The module view name or names list is also saved, or persisted in the workspace metadata for each module so that all subsequent populates use the same view. The documentation refers to a view saved in the metadata as a "persistent module view" because it persists through subsequent populates rather than needed to be specified with each command.

If a persistent module view has been set on a workspace module, any sub-modules subsequently populated use the persistent module view already defined for parent module.

Tip: Since populate calls the Checkout Access Control, you can write an Access Control filter to cause populate to fail if no module view is specified or tie users to specific module views.

Notes:

- If none of the specified module views exist on the server, DesignSync issues a warning and the populate command runs as if no view were specified. If, in a list of module views, one or more views exists, and one or more views does not exist, the populate command silently ignores the non-existent view(s).
- When the persistent module view set on the workspace is changed, the subsequent populate is a full populate. For more information on changing or clearing the persistent view, see the setview command.

-xtras (Module-based)

-xtras <list>

List of command line options to pass to the external module change management system. Any options specified with the -xtras option are sent verbatim, with no processing by the populate command, to the Tcl script that defines the external module change management system.

RETURN VALUE

```
In dss/dssc mode, you cannot operate on return values, so the return
value is irrelevant.

In stcl/stclc mode, two lists are returned, where the first list
is non-empty if at least one object was successfully processed,
and the second list is non-empty if at least one object failed.

Notes:
 - "successfully processed" may not mean "successfully populated".
   For example, a populate of an object that you already have in
    your work area is considered a success even though no checkout
   occurs.
 - Scripts should only test for non-empty lists to determine
   success or failure. The actual content of the non-empty
   lists might change in subsequent releases.
 - If all objects fail, an exception occurs (the return value is
   thrown, not returned).
 - Objects reported as "excluded by filter," may have been excluded
   either with the -filter option (for modules) or with the -exclude
   option (for any DesignSync objects.)
```

SEE ALSO

```
caching, ci, co, command defaults, localversion, remove, retire,
selectors, setselector, setvault, setview, swap, url contents
```

EXAMPLES

- Example of Populating a Module (Module-based)
- Example of Populating a Specific Module Member (Module-based)
- Example of Populating a Module with a Static Selector (Module-based)
- Example of Populating a Module Using Version-Extended Naming (Module-based)
- Example of Creating a Module Cache (Module-based)
- Example of Populating an Mcache Link (Module-based)
- Example of Populating a Module View (Module-based)
- Example of Specifying a Hierarchical Hreffilter (Module-based)
- Example of Merge Across Branches (Module-based)
- Example of Creating a new work area from a DesignSync vault (File-based)
- Example of Creating a New Work Area from a DesignSync Vault Branch (File-based)
- Example of Updating an Existing Workspace with a Full Populate (File-based)
- Example of Updating the State of Objects in the Workspace (File-based)
- Example of Performing a Merge into a Workspace (File-based)
- Example of Replacing Modified Files with the Server Versions (File-based)

Example of Populating a Module (Module-based)

```
The following example shows how to populate module Chip in the
workspace directory ~/chip.
For an initial populate, provide the server URL of the module:

stcl> pop sync://guaraldi:30077/Modules/Chip

This creates the Chip module with the current directory
as the base directory:

Beginning populate operation...

Making Module with
  Base Dir = /home/karen/chip
  Name = Chip
  URL = sync://guaraldi:30077/Modules/Chip
  Selector = Trunk:Latest

Created Module with instname Chip%1

Populating objects in Module Chip%1 with Base Dir /home/karen/chip...

/chip/makefile:  Success - Checked Out  version: 1.1
/DOC/Chip.doc:  Success - Checked Out  version: 1.1
/chip/verilog/chip.v:  Success - Checked Out  version: 1.1

Chip%1: Version of module in workspace updated to 1.2

Finished populate of Module Chip%1 with Base Dir  /home/karen/chip

Finished populate operation...

{Objects succeeded (3)} {}

When you next update your work area using the populate command,
you can supply the workspace module name or the workspace folder
name.  In the following example the workspace folder name is
supplied, and there have been no changes since the last populate:

 stcl> pop -recursive ~/chip
  Beginning populate operation at Thu Apr 19 02:16:31 PM EDT 2007...



  Populating objects in Module          Chip%1
                       Base Directory  /home/karen/chip
                       Without href recursion



  Chip%1 : Version of module in workspace retained as 1.2

  Finished populate of Module Chip%1 with base directory
  /home/karen/chip

  Finished populate operation.

   {} {}
```

Example of Populating a Specific Module Member (Module-based)

```
 The following is an example of fetching a specific version of a
 module member:

  stcl> pop -version 1.4 File1.txt

  Populating objects in Module          JitaMod1%0
                      Base Directory  /home/tachatterjee/JitaMOD
                      Without href recursion


  Fetching contents from selector '1.4', module version '1.4'

  Total data to transfer: 0 Kbytes, 1 files, 0 collections
  Progress: 0 Kbytes, 1 files, 0 collections, 100.0% complete
  /File1.txt: Success - Checked Out  version: 1.3

  Finished populate operation...

 This fetches the version of the file File1.txt contained in
 version 1.4 of the module.
```

Example of Populating a Module with a Static Selector (Module-based)

```
 The following example shows the messages you receive when you
 populate a static selector into a workspace.

 dss>  populate -recursive -version Gold Chip-R419%0
  Beginning populate operation at Fri Oct 28 12:41:08 Eastern Daylight
  Time 2016...

  Setting Selector [Gold] on workspace module
  c:\workspaces\ChipDev419\chip\Chip-R419%0
  WARNING: Chip-R419%0: Changing the selector to a static value (Gold).
  You will not be able to check in module or member modifications.

  Selector on module c:\workspaces\ChipDev419\chip\Chip-R419%0 was
  modified.

  Populating objects in Module          Chip-R419%0
                       Base Directory  c:\workspaces\ChipDev419\chip
                       With href recursion


  Fetching contents from selector 'Gold', module version '1.5.1.1'
...
  Finished populate operation.

  ##### WARNINGS and FAILURES LISTING #####
  #
  # WARNING: Chip-R419%0: Changing the selector to a static value
```

```
#(Gold).
# You will not be able to check in module or member modifications.
#
########################################

 {Objects succeeded (6)} {Objects failed (0)}
```

Example of Populating a Module Using Version-Extended Naming (Module-based)

```
The following example shows how to fetch a specific version of a
module using a version-extended name.

In this example, the latest version of the file is 1.5. You can
do a vhistory to determine which version of the file you want to
fetch.

To fetch version 1.2 of the file:

 stcl> pop "File1.txt;1.2"

 Beginning Check out operation...

 Checking out: File1.txt          : Success - Fetched  version: 1.2

 Checkout operation finished.


 Finished populate operation...
```

Example of Creating a Module Cache (Module-based)

```
The following example shows how to populate a module cache using the
-share option to create a copy of the module in a centralized
location.

Note:  The module cache directory must be writable by the
creator/owner of the module cache, but not by the users of the module
cache.

stcl> populate -share -
```

Example of Populating an Mcache Link (Module-based)

```
The following example shows how to populate module Chip
using the -mcachepaths option to fetch contents from the module
cache named 'designs' located in the mcacheDir directory.

  stcl> populate -get -recursive -hrefmode static
   -path /home/rsmith/MyModules/designs -mcachemode link -mcachepaths
  /home/mcacheDir/  sync://srv2.ABCo.com:2647/Modules/Chip/
```

```
  Beginning populate operation at Mon Jun 23 10:36:43 AM EDT 2008...

  sync://srv2.ABCo.com:2647/Modules/Chip/: : Created mcache
  symlink /home/rsmith/MyModules/designs.

  Creating Module Instance 'Chip%1' with base directory
  '/home/rsmith/MyModules/designs'

  Finished populate operation.

   {Objects succeeded (1)} {}
```

Note: Any existing workspace content will not be replaced with module cache links. To replace workspace content you must first remove from the workspace those configurations to be replaced. Use the 'rmfolder -recursive' command on the configuration base directory, or specify a non-existent directory for the -path option to create a new directory for the module cache links.

Example of Populating a Module View (Module-based)

This example shows populating a workspace with a module view list; specifically the the RTL and DOC Module Views.

```
stcl> populate -get -view RTL,DOC -path ./Chip sync://
srv2.ABCo.com:2647/Modules/Chip

Beginning populate operation at Fri May 06 02:04:38 PM EDT 2011...

Populating module instance with

Base Directory = /users/larry/MyModules/Chip
Name = Chip
URL = sync:// srv2.ABCo.com:2647/Modules/Chip
Selector = Trunk:
Instance Name = Chip%2
Metadata Root = / users/larry/MyModules
View(s) = RTL,DOC

Recursive Mode = Without href recursion

Fetching contents from selector 'Trunk:', module version '1.9'
Total data to transfer: 1 Kbytes (estimate), 5 file(s), 0 collection(s)

Progress - from local cache: 0 Kbytes, 0 file(s), 0 collection(s)
Progress - from server: 1 Kbytes, 5 file(s), 0 collection(s), 100.0%
complete

Chip%2/makefile : Success - Checked out version: 1.2
Chip%2/README : Success - Checked out version: 1.3
Chip%2/doc/chip.html : Success - Checked out version: 1.2
Chip%2/doc/chip.doc : Success - Checked out version: 1.2
Chip%2/verilog/chip.v : Success - Checked out version: 1.5
```

```
 Chip%2/verilog/chip_inc.v : Success - Checked out version: 1.3

 Chip%2 : Version of module in workspace updated to 1.9

 Finished populate of Module Chip%2 with base directory
 /users/larry/MyModules/Chip

 Time spent: 0.2 seconds, transferred 1 Kbytes, copied from local
 cache 0 Kbytes, average data rate 4.9 Kb/sec

 Finished populate operation.

 {Objects succeeded (5)} {}
```

Example of Specifying a Hierarchical Hreffilter (Module-based)

```
 This example shows an initial populate using a hierarchical href
 filter to exclude the /BIN module from the workspace when it appears
 beneath the /JRE module. In this example, the module hierarchy is set
 up like this:
 NZ214 <- ROM <- JRE <- BIN
 With NZ214 being the top-level Chip design module.

 Note: Whenever you use the -hreffilter option, you must populate
 recursively.

 dss> populate -recursive -retain -full -hreffilter JRE/BIN
 sync://serv1.ABCo.com:2647/Modules/Chip/NZ214

  Beginning populate operation at Wed Dec 11 13:24:31 Eastern Standard
 Time 2013...

  Populating module instance with
     Base Directory  = c:\workspaces\V6R2014x\chipDesign
     Name            = NZ214
     URL             = sync://serv1.ABCo.com:2647/Modules/Chip/NZ214
     Selector        = Trunk:
     Instance Name   = NZ214%1
     Metadata Root   = c:\workspaces\V6R2014x
     Recursive Mode  = With href recursion


  Fetching contents from selector 'Trunk:', module version '1.3'

  Total data to transfer: 0 Kbytes (estimate), 6 file(s), 0 collection(s)
  Progress - from local cache: 0 Kbytes, 0 file(s), 0 collection(s)
  Progress - from local cache: 0 Kbytes, 0 file(s), 0 collection(s)

  Progress - from server: 0 Kbytes, 0 file(s), 0 collection(s), 0.0% complete
  Progress - from server: 1 Kbytes, 6 file(s), 0 collection(s), 100.0%
complete

  NZ214%1\chip.ver : Success - Checked out version: 1.1
 ...
```

```
  Creating sub module instance 'ROM%1' with base directory
 'c:\workspaces\V6R2014x\chipDesign\ROM'

 Finished populate of Module NZ214%1 with base directory
 c:\workspaces\V6R2014x\chipDesign

 Time spent: 0.3 seconds, transferred 1 Kbytes, copied from local cache 0
Kbytes, average data rate 3.4 Kb/sec

  ======================================================

  Populating sub module instance with
     Base Directory  = c:\workspaces\V6R2014x\chipDesign\ROM
     Name            = ROM
 ...
  Creating sub module instance 'JRE%0' with base directory
     'c:\workspaces\V6R2014x\chipDesign\ROM\JRE'

 Finished populate of Module ROM%1 with base directory
 c:\workspaces\V6R2014x\chipDesign\ROM

 Time spent: 0.0 seconds, transferred 0 Kbytes, copied from local
 cache 0 Kbytes, average data rate 0.0 Kb/sec

  ======================================================

  Populating sub module instance with
     Base Directory  = c:\workspaces\V6R2014x\chipDesign\ROM\JRE
 ...
  JRE%0 : Version of module in workspace updated to 1.2

  BIN :   Sub Module Excluded by Hierarchical Filter
  Finished populate of Module JRE%0 with base directory
  c:\workspaces\V6R2014x\chipDesign\ROM\JRE

  Time spent: 0.0 seconds, transferred 0 Kbytes, copied from local
  cache 0 Kbytes, average data rate 0.0 Kb/sec


  Finished populate operation.

  {Objects succeeded (8)} {}
```

Example of Merge Across Branches (Module-based)

```
 This example shows a simple module merge across branches.  After you
 perform the merge, you must check in your changes to apply the merge
 changes to the modules.

 dss> pop -merge -overlay Branch: ROM%1
  Beginning populate operation at Tue Apr 10 01:55:24 PM EDT 2007...
```

```
Populating objects in Module          ROM%1
                      Base Directory  /home/rsmith/MyModules/rom
                      Without href recursion


Fetching contents from selector 'Branch:', module version '1.3.1.3'

Merging with Version: 1.3.1.3
Common Ancestor is Version: 1.3

========================================================================
Step 1: Identifying items to be merged and conflict situations
========================================================================

/romMain.c : member will be fetched from merged version and
      added to workspace version on checkin.
      Use 'ls -merged added' to identify members added by merge.
/rom.v : conflict - different member in merge version found at same natural
       path in workspace version. Cannot fetch member or merge contents
       with member from merge version; it will be skipped. If member from
       merge version is desired, remove or move member on workspace
       branch and then re-populate with overlay from merge version.
/rom.v : Natural path different on merge version and workspace version.
      Contents will be merged, if required.
/rom.doc : No merge required.
/doc/rom.doc : No merge required.

========================================================================
Step 2: Transferring data for any items to be fetched into the
workspace
========================================================================

Total data to transfer: 0 Kbytes (estimate), 1 file(s), 0 collection(s)
Progress: 0 Kbytes, 0 file(s), 0 collection(s), 0.0% complete
Progress: 1 Kbytes, 1 file(s), 0 collection(s), 100.0% complete


============================================================================
Step 3: Merging file contents as required into the workspace
============================================================================


Beginning Check out operation...

Checking out: rom.v                          : Success - Version
1.1.1.1 has replaced version 1.1.
Checking out: rom.c                          : Success - Version
1.1.1.1 has replaced version 1.1.

Checkout operation finished.


========================================================================
Step 4: Updating files fetched into the workspace
========================================================================

/romMain.c : Success - Version 1.1 fetched
```

```
 ROM%1 : Version of module in workspace not updated (Due to overlay
 operation).


 ======================================================================
 Step 5: Comparing hrefs for the workspace version and merge version:
 ======================================================================
     No hrefs present in workspace version
     No hrefs present in merge version


 Finished populate of Module ROM%1 with base directory
 /home/rsmith/MyModules/rom

 Time spent: 0.2 seconds, transferred 1 Kbytes, average data rate 4.3 Kb/sec

 Finished populate operation.

  {Objects succeeded (3)} {}
```

After the populate has completed, run ci to create the new module version with the merge changes.

```
dss> ci -comment "Incorporating changes on Branch:" ROM%1
   Beginning Check in operation...


 Checking in objects in module ROM%1

 Total data to transfer: 1 Kbytes (estimate), 3 file(s), 0 collection(s)
 Progress: 0 Kbytes, 0 file(s), 0 collection(s), 0.0% complete
 Progress: 2 Kbytes, 3 file(s), 0 collection(s), 100.0% complete

 Checking in: /rom.c                        Success - New version: 1.2
 Checking in: /rom.v                        Success - New version: 1.2
 Checking in: /romMain.c                    Success - New version: 1.1.1.1

 ROM%1: Version of module in workspace updated to 1.5

 Finished checkin of Module ROM%1, Created Version 1.5

 Time spent: 0.7 seconds, transferred 2 Kbytes, average data rate 2.8 Kb/sec
 Checking in: /doc/rom.doc                  : Success - No new
 version created. Lock Removed.

 Checkin operation finished.

  {Objects succeeded (4)} {}
```

After the checkin has created the new module version, you can create a merge edge to store a record of the changes.

```
 dss> mkedge ROM%1
 Edge from 1.3.1.3 to 1.5 for module
 sync://srv2.ABCo.com:2647/Modules/ROM created successfully.
```

Example of Creating a new work area from a DesignSync vault (File-based)

The following example creates a new work area containing unlocked copies of every object in the vault hierarchy:

```
dss> scd /home/tgoss/Projects/Asic
dss> setvault sync://myhost.myco.com:2647/Projects/Asic .
dss> populate -recursive -get
```

Because -version is not specified, the persistent selector list of the current folder determines what versions to populate. The local Asic folder has not had a 'setselector' command applied to it or any parent folder, so the default persistent selector list is 'Trunk'. By default, DesignSync performs an incremental populate of the Latest versions on the specified branch (Trunk). Note that this operation does not fetch objects whose 'Trunk' branch is retired.

Example of Creating a New Work Area from a DesignSync Vault Branch (File-based)

The following example differs from the previous example in that the work area is for the Rel2.1 branch, not Trunk, and the work area contains links to a cache directory instead of local copies:

```
dss> scd /home/tgoss/Projects/Asic
dss> setvault sync://myhost.myco.com:2647/Projects/Asic@Rel2.1:Latest .
dss> populate -recursive -share
```

Example of Updating an Existing Workspace with a Full Populate (File-based)

The following example performs a full (nonincremental) recursive populate on the current folder, fetching unlocked copies of files for updated objects. Note that the states of objects that are not updated DO NOT change.

```
dss> populate -recursive -full -get
```

Example of Updating the State of Objects in the Workspace (File-based)

By default, the states of up-to-date objects do not change during a populate operation. The following example updates the states of the objects that are up-to-date, allowing you to unify the states of all objects in your work area. The -unifystate option causes DesignSync to perform a full populate rather than an incremental populate.

```
dss> populate -recursive -unifystate -get
```

Example of Performing a Merge into a Workspace (File-based)

The following example merges Latest versions from the current branch into the local versions. You perform this operation when your team uses the merging (nonlocking) work model and you and other team members have been modifying the same objects. It is more common to use the 'co -merge' command to operate on just those objects you want to check in.

```
dss> populate -merge
```

Note that the merge operation fetches from the branch specified by the folder's persistent selector list, not from the current branch. However, these two branches are typically the same unless you have changed the persistent selector list with the setselector command. In this case, you would be merging across branches instead of from the same branch. This method for merging between two branches is not recommended; use the -overlay option.

The following example merges one branch (Dev) into another (Main). This operation is typically performed by a release engineer who manages the project vault. The work area is first populated with the Latest versions from 'Main'. Then the Latest versions from Dev are merged into the local versions. The -overlay option indicates that after the operation, the current branch and version information (as stored in local metadata) should be unchanged. Following the merge and after any merge conflicts are resolved, a check-in operation checks the merged version into 'Main'.

```
dss> url selector .
Main:Latest
dss> populate -recursive
dss> populate -recursive -merge -overlay Dev:Latest
  [Resolve any merge conflicts]
dss> ci -recursive -keep .
```

Example of Replacing Modified Files with the Server Versions (File-based)

This example shows use of the populate operation that deletes local versions.

Note: The DesignSync Milkway integration has been deprecated. This example is meant to be used only as a reference.

Mike checks out the Milkyway collection object top_design.sync.mw, which fetches local version 4 of that object to his workspace. He modifies the object and creates local version 5. Then he checks in top_design.sync.mw. The check-in operation does not remove local versions, so Mike now has local version 5 (unmodified) and local version 4 in his workspace. (Note: Because the checkin removes local version 4's link to with the original check-out operation of top_design, DesignSync now considers local version 4 to be modified.)

Ben checks out top_design.sync.mw (local version 5). He creates local version 6 and checks the object in.

Mike does some work on top_design, which creates local versions 6, 7, and 8 in his workspace. Then he decides to use Ben's version of the top_design object instead.

Mike uses populate to fetch the latest versions of Milkyway collection objects to his workspace. He doesn't want to save his local versions of the object, so he uses the '-savelocal delete' option to delete local versions other than the local version being fetched. In addition, he uses the -force option. (Because he created local versions 6, 7, and 8 of top_design in his workspace, DesignSync considers the top_design object to be locally modified and by default the populate operation does not overwrite locally modified objects. To successfully check out top_design, Mike must use '-force'.)

```
stcl> cd /home/tjones/top_design_library
stcl> populate -savelocal delete -force
```

Before fetching top_design.sync.mw from the vault, the populate operation first deletes all local versions that are unmodified. So the populate operation deletes Mike's local version 6 because that was the version originally fetched and its files are unmodified.

Because Mike specified the -force option, the populate also deletes Mike's local version 8 (the current local version containing modified data for the object).

Because Mike specified '-savelocal delete', the populate operation deletes local version 7, which is not in the vault and is not the modified data Mike agreed to delete when he specified '-force'. If Mike specified '-savelocal save', DesignSync would save local version 7. Local version 4 is also deleted.

Finally, Mike's populate operation fetches the top_design object (Ben's local version 6) from the vault.

Mike continues to modify the top_design object, creating local version 7, which he checks in.

Ben has local versions 5 and 6 in his workspace. He populates his workspace containing the top_design collection object (local version 7), specifying '-savelocal fail'. The populate operation removes local version 6 from his workspace because it is unmodified. The operation saves local version 5 even though it is modified. (Ben's checkin of local version 6 removed local version 5's link to with the original checkout of top_design, so DesignSync now considers local version 5 to be modified.) The populate also takes place despite the fact that Ben specified '-savelocal fail'. The populate operation takes this action because local version 5 has a number lower than the local version being fetched. If Ben had instead specified '-savelocal delete', the populate operation would delete local version 5.

tag

tag Command

NAME

```
tag                  - Assigns a tag to a version or a branch
```

DESCRIPTION


```
This command assigns a symbolic name, called a tag, to a version
(version tag) or branch (branch tag). You also use this command to
move (-replace) or remove (-delete) existing tags.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

This command supports Enterprise Design Synchronization. For more
information on Enterprise Design Synchronization, see the Enterprise
Design Administration User's Guide.

This command supports the command defaults system.
```

Working with Tags

```
Tagging a set of versions creates a group of objects, sometimes
called a configuration, that is a representation of your design files
that correspond to a known state, such as a development or release
milestone. For example, you might tag the current versions of your
design files 'Alpha' when you have reached the Alpha milestone.

Once you have tagged your set of versions, the tag can be used as a
selector to identify what objects commands operate on. For example,
you might specify 'populate -version Gold' to populate all versions
```

```
that are tagged 'Gold' (the 'Gold' configuration). See the
"selectors" help topic for more information on selectors.

Versions and branches can have more than one tag assigned to
them. For example, an object that did not change between releases
might have both 'rel2.1' and 'rel2.2' applied to the same version.

Note: If you tag a version with a tag that already exists on that
version, the system will respond with a 'success' message.
```

Branch Tags Versus Version Tags

```
Branch tags and version tags share the same name space. To
distinguish version selectors from branch selectors, you
append ':<versiontag>' to the branch name; for example,
'Gold:Latest' is a valid branch selector.  You can leave off the
'Latest' keyword as shorthand; for example, 'Gold:' is equivalent
to 'Gold:Latest'. The selector 'Trunk' is also a valid branch
selector; 'Trunk' is a shorthand selector for 'Trunk:Latest'.

You cannot assign the same tag name to both a version and a branch
of the same object. For example, a file called 'top.v' cannot have
both a version tagged 'Gold' and a branch tagged 'Gold'. However,
'top.v' can have a version tagged 'Gold' while another file, 'alu.v',
can have a branch tagged 'Gold'.

Consider adopting a consistent naming convention for branch
and version tags to reduce confusion. For example, you might have a
policy that branch tags always begin with an initial uppercase
letter ('Rel2.1', for example) whereas version tags do not ('gold',
for example).

If the selector identifies a version, DesignSync resolves the
selector to both the object's version number and branch number.
For example, if version 1.2.1.3 is tagged 'gold', DesignSync
resolves 'gold' as both version 1.2.1.3 and branch 1.2.1.
A version selector only resolves if the object has a version
tag of the same name; it does not resolve if the tag is a branch
tag. For example, if branch 1.2.1 is tagged 'RelA', and the
latest version on that branch is 1.2.1.3, then DesignSync
resolves 'RelA:Latest' as version 1.2.1.3; however, DesignSync
does not resolve selector 'RelA' at all, because there is no
version tag of that name.
```

Tagging Modules (Module-based)

```
The tag operation for modules tags versions or branches of the module
in the vault, not the local copies of objects in your work
area.
```

Module Snapshots (Module-based)

Module snapshots are a collection of versionable module members that are tagged from a workspace. When you tag a set of member versions in a workspace, you create a new "snapshot" branch on the server. Using a branch allows you to maintain a snapshot as a versionable object, updating tags and hierarchical references as needed.

Module snapshots allow you to capture a subset of a module workspace at any given moment in time, and recreate it. This can be useful to preserve a specific set of files for testing or releasing that set of files without interrupting the normal development workflow.

When you create a module snapshot, DesignSync creates a special snapshot branch for the module. When you create the snapshot, you provide a tag name; the module branch is created with the name SNAPSHOT_<tag_name>. The specific snapshot version is <tag_name>.

Operations on tagged module snapshots are always workspace-centric. This means the operations occur on the objects loaded in the workspace. If a folder is specified with recursion, the operation traverses the folder.

The module snapshot is restricted to a single module, however you can update multiple module snapshots in a single tag operation. You can restrict a tag operation to a single module by using the -modulecontext switch to select the desired module.

The module snapshot operations are atomic with respect to the server. In order to execute the tag operation, all objects within a module must be processed successfully. If any object fails the entire operation fails for that module. For example, if you tag module members in your workspace belonging to different modules and you do not have tag access for one of the modules or module members, the tag fails for that module only. The other modules, assuming no other errors within them, are updated successfully.

The module snapshot operations are not atomic with respect to the workspace. For example, if you have a moved, removed, or added a file that has not been checked in, it does not cause the entire tag operation to fail. You receive an error message for any individual workspace object that failed, and the operation itself succeeds.

Hierarchical references within module snapshots must be manually added or removed. DesignSync does not automatically include hierarchical references already in the workspace in a new module snapshot, nor does it update hierarchical references in the snapshot when the snapshot is versioned by adding or removing tags. After the snapshot has been created, you can add the desired hierarchical references to the snapshot, and update, remove, or add new hierarchical references as needed.

Operations that can create a module version with structural or content changes, such as add, remove, checkin, mvmember, rollback, and populate with the -lock option, are not allowed with module snapshots. These snapshots are intended to be used as is, with

```
content frozen.  The only operations allowed are addhref, rmhref,
edithrefs, and tag operations (adding, removing, or moving tag names
from module members). This allows you to create the perfect,
immutable, test or release version.
```

Tag Name Syntax (Module-based)

```
The first argument to the 'tag' command is the tag name.
Tag names:
  - Can contain letters, numbers, underscores (_), periods (.),
    hyphens (-), and forward slashes (/). All other characters,
    including whitespace, are prohibited.
  - Cannot start with a number and consist solely of numbers
    and embedded periods (for example, 5, 1.5, or 44.33.22),
    because there would be ambiguity between the tag name and
    version/branch dot-numeric identifiers.
  - Cannot be any of the following reserved, case-insensitive keywords:
    Latest, LatestFetchable, VaultLatest, VaultDate, After,
    VaultAfter, Current, Date, Auto, Base, Next, Prev, Previous,
    Noon, Orig, Original, Upcoming, SyncBud, SyncBranch, SyncDeleted.
    Also, avoid using tag names starting with 'Sync' (case-insensitive),
    because Synchronicity may define new keywords in the future
    using that naming convention.


    Note: The Connected Software and Connected Semiconductor apps do
    not support the use of forward slash (/) in Tag names.

The 'Latest' reserved keyword is of particular importance. 'Latest'
is always associated with the most recent (highest numbered)
version of a design object on a given branch. Although not actually a
tag, you can generally specify 'Latest' as you would a user-defined
version tag. Note that the default command behavior in many cases
is to operate on the latest version on the current or specified
branch, so you typically do not need to specify 'Latest'. See the
"selectors" help topic for more details on selectors, including the
use of 'Latest'.

The 'Trunk' tag, although not a reserved keyword, has special
significance for DesignSync. By default, DesignSync tags branch 1
as 'Trunk' when you initially check in a design object. Because
'Trunk' is a tag (shorthand for 'Trunk:Latest'), you can move
or delete it, although doing so is not recommended.  Due to this
special significance, the 'Trunk' tag is always expected to be a
branch tag, and you cannot add this as a version tag. For example,
you can specify 'tag -branch 1 Trunk myfile', but you cannot
specify 'tag -version 1.1 Trunk myfile'.
```

Determining the Objects to be Tagged (Module-based)

Each object argument to the 'tag' command can be:

- A module, specified explicitly as a server module URL, in this format: sync://<machine>:<port>/Modules/<category>/<module_name>;<selector>

- A module, specified as a workspace module instance. This behaves identically to specifying the module explicitly as a server module URL. It does not tag the local versions in the workspace, nor does it create a module snapshot.

  Note: When used on workspace module instance, the -modified option is ignored, since the tagged object is the last server module version populated into the workspace, not the locally modified files.

- Module members and module member folders can be tagged explicitly as part of a module snapshot. The module snapshot is a tagged configuration presented as a side branch that allows for hierarchical reference and tag updates within the snapshot, but does not allow content changes to the module members or structural changes to the module.

  Note: There is a limitation when -modulecontext is used to restrict the tag to members of a particular module and wildcarding is used to specify members to tag. If a module member within the directory cone matches the tag, but is not part of the specified module, and you cannot tag that member the operation fails. If you can tag the member, the operation succeeds, but the member is not part of the tagged module snapshot. It is excluded because it is not part of the specified module.

- A branch object. The latest version on the specified branch is tagged unless you specify the -branch option, in which case a branch tag is applied to the branch object you specified -- the argument to the -branch option is ignored.

Note: Tag supports both filter and exclude which can affect which objects available for tagging.

Using Tags on Module Versions (Module-based)

To manage the development of modules, you can use tags to indicate that a module is ready to be released to team members. You use the -immutable option to apply a tag that cannot be moved. Your team can implement a methodology by which a release engineer applies an immutable, or fixed, tag to a design module when the module has reached a particular quality threshold. Your Synchronicity administrator can enforce the methodology by setting access controls to control the addition or removal of immutable tags. See Access Control Guide: "Access Controls for Tagging" for details. See also the -[im]mutable option description below to learn how to add, move, or delete immutable and mutable tags.

You can also use a module snapshot to manage an immutable release

version. For more information on module snapshots, see the Module Snapshots section.

When you tag a module hierarchy, using the -recursive tag, you are tagging the selected module and the referenced submodule in static mode using the specified module version on the server, preserving the exact versions of all the files you were working with, regardless of whether the module was specified as a server module URL, or a workspace module.

Note: If a module contains hierarchical references to different versions of the same module, only the first version found is tagged and DesignSync will return an error explaining the situation. If multiple module arguments are specified, each hierarchy of all specified modules is expanded prior to processing, and any duplicate modules with different versions fail with an error.

Interaction with Legacy Modules (Legacy-based)

Important: Legacy modules are modules generated prior to the 5.0 DesignSync release. It is strongly suggested that you upgrade your legacy modules using the upgrade command. See the "Working with Legacy Modules" book in DesignSync Data Manager User's Guide for more information about legacy modules.

Prior to 5.0, modules were managed with configurations. Modules no longer require these configurations. The following description of configurations helps illustrate the use of legacy modules only.

If a recursive tag operation encounters a vault folder on the SyncServer that is configuration-mapped to another vault folder (using DesignSync REFERENCEs), the tag operation's behavior depends on how you populated the configuration-mapped folder to your work area:

- If you populated with a static tag (for example, a version tag such as -version Gold), when you use 'tag -recursive', the tag operation creates a new configuration map on the SyncServer instead of tagging the objects in the vault folder. However, if the DesignSync REFERENCE is to a vault folder that has space characters in its name, its configuration map attempt will fail.

- If you populated with a dynamic tag (for example, a branch tag such as -version Test1Branch:Latest), when you use 'tag -recursive', the tag operation recurses into the local folder and uses the objects in that folder to determine which objects to tag in the vault.

For more information about populating a configuration-mapped vault folder, see the "Interaction with ProjectSync: Configuration Mapping" section of the populate command.

If a recursive tag operation encounters a legacy module

```
configuration, the operation does not create a new hierarchical
reference. Instead, the tag operation recurses into the local folder
and uses the objects in the local folder to determine which objects
to tag in the corresponding vault folder.
```

Tagging Files-Based DesignSync Objects (File-based)

```
Tags for DesignSync vaults use the object versions in your work area
to determine the appropriate version or branch to tag in the
vault. Once a tag operation has completed, the new tags are visible
to other users of the vault.  If the version you want to tag is not
the version in your workspace, use the -version option to specify the
correct version or branch to tag.

If you want to tag a locally modified DesignSync object, you must
specify the -modified option.

Note:  If you tag a directory that includes unmanaged objects, the
tag operation does not return an error for the unmanaged objects, but
rather fails silently.
```

Tag Name Syntax (File-based)

```
The first argument to the 'tag' command is the tag name.
Tag names:
  - Can contain letters, numbers, underscores (_), periods (.),
    hyphens (-), and forward slashes (/). All other characters,
    including whitespace, are prohibited.
  - Cannot start with a number and consist solely of numbers
    and embedded periods (for example, 5, 1.5, or 44.33.22),
    because there would be ambiguity between the tag name and
    version/branch dot-numeric identifiers.
  - Cannot be any of the following reserved, case-insensitive keywords:
    Latest, LatestFetchable, VaultLatest, VaultDate, After,
    VaultAfter, Current, Date, Auto, Base, Next, Prev, Previous,
    Noon, Orig, Original, Upcoming, SyncBud, SyncBranch, SyncDeleted.
    Also, avoid using tag names starting with 'Sync' (case-insensitive),
    because Synchronicity may define new keywords in the future
    using that naming convention.

    Notes:
    o The Connected Software and Connected Semiconductor apps do
      not support the use of forward slash (/) in Tag names.
    o DesignSync vaults and legacy modules have an additional
      restriction: tag or branch names cannot end in --R.

The 'Latest' reserved keyword is of particular importance. 'Latest'
is always associated with the most recent (highest numbered)
version of a design object on a given branch. Although not actually a
tag, you can generally specify 'Latest' as you would a user-defined
version tag. Note that the default command behavior in many cases
is to operate on the latest version on the current or specified
```

branch, so you typically do not need to specify 'Latest'. See the "selectors" help topic for more details on selectors, including the use of 'Latest'.

The 'Trunk' tag, although not a reserved keyword, has special significance for DesignSync. By default, DesignSync tags branch 1 as 'Trunk' when you initially check in a design object. Because 'Trunk' is a tag (shorthand for 'Trunk:Latest'), you can move or delete it, although doing so is not recommended. Due to this special significance, the 'Trunk' tag is always expected to be a branch tag, and you cannot add this as a version tag. For example, you can specify 'tag -branch 1 Trunk myfile', but you cannot specify 'tag -version 1.1 Trunk myfile'.

Determining the Objects to be Tagged (File-based)

Each object argument to the 'tag' command can be:

- A local managed object (file or collection object). By default, the current version in your work area is tagged unless you specify the -branch option or -version option.

  Note: If you specify a collection member as the object to be operated on, DesignSync skips the object and warns that the object is not versionable. If DesignSync attempts to operate on a collection member specified implicitly (through the use of wildcards or a recursive operation), DesignSync silently skips the object. You can change this behavior by using the SyncAdmin "Map operations on collection members to owner" setting. If you select this setting and DesignSync attempts to operate on a collection member during a revision control operation, DesignSync determines the member's owner collection and operates on the collection as a whole.

- A folder, which is useful only for recursive tagging. You can specify either a local folder (a folder in your work area) or a vault folder:

  - If you specify a local folder, the tag operation uses the objects in that local folder to determine which objects to tag in the corresponding vault folder. If the tag operation encounters a module, it does not tag the module. To tag a module, specify the server URL of the module. If the tag operation encounters a legacy module, tag handles the module as a customary DesignSync workspace, tagging the objects and continuing to traverse the hierarchy according to the -recursive option selected. Note, however, that the tag command will not stop at submodule boundaries. (The -nomodulerecursive option is no longer supported.)

  - If you specify a vault folder, the tag operation tags each object in the vault folder. Note: If you specify a vault folder for a recursive tag operation, the operation does not create new configuration maps for legacy DesignSync REFERENCEs.

- A version object, although to identify a particular version, you typically specify a local object and the -version option. The -version option is ignored if you specify a version object.

- A branch object. The latest version on the specified branch is tagged unless you specify the -branch option, in which case a branch tag is applied to the branch object you specified -- the argument to the -branch option is ignored.

Note: Tag supports both filter and exclude which can affect which objects available for tagging.

Interaction with Objects from a Mirror (File-based)

If a work area contains a link to a mirror, the tag operation uses the version that resides in the mirror directory to determine which object version to tag in the vault, even though the version in the mirror directory may not be the Latest version in the vault. For example, if your work area contains a link to version 1.3 of fileA in the mirror directory, a tag operation tags version 1.3 in the vault, even though fileA's 'Latest' version in the vault is 1.4.

Notes:

- The tag operation considers an object that is a link to a mirror as unmodified and does not fail for that object. The same is true for members of collections: if all members of a collection are symbolic links, then the collection is not considered by the tag operation as modified, even if a member symbolic link was deleted.

- Synchronicity does not recommend tagging a work area that contains links to a mirror directory. A mirror directory is updated constantly; if you tag objects while the mirror's objects are changing, the result may be a configuration different from the one you intended.

SYNOPSIS

```
tag [-branch <branch> | -branch auto(<branch>) | -[no]delete |
     -[no]replace |  -version <selector>] [-[no]comment <text>]
    [-exclude <object>[,<object>,...]] [-filter <object>[,<object>]]
    [-[no]modified] [-modulecontext <context>] [-[im]mutable]
    [-[no]recursive] [-report <mode>] [-[no]selected]
    [-trigarg <arg>] [-warn <mode>] [-xtras <xtras>] [--] <tagname>
    [<argument> [<argument> ...]]
```

ARGUMENTS

- Server Module (Module-based)
- Module Folder (Module-based)
- Module Member (Module-based)
- Workspace Module (Module-based)
- External Module (Module-based)
- DesignSync Object (File-based)
- DesignSync Folder (File-based)

```
The tag command accepts a <tagname> followed by multiple arguments
on which to apply the tag. See "Tag Name Syntax" above for the
allowable values for the tagname.

Specify one or more of the following arguments:
```

Server Module (Module-based)

```
<server module>     Tags the specified module in its vault.
                    Specify the module's server URL in the format:

                    sync://<machine>:<port>/Modules/<category>/
                    <module_name>;<selector>

                    If the specified server module is a legacy
                    module, the operation does not create new
                    configuration maps for DesignSync REFERENCES or
                    follow hierarchical references.
```

Module Folder (Module-based)

```
<module folder>     When a module folder is selected, the operation
                    creates or updates the appropriate snapshot
                    branch. Module folders are only valid arguments
                    when using module snapshots.  You must specify the
                    module folder as a workspace objects, for
                    example:
                    ./<folder_name>
                    <Module_Instance_Name>/<folder_name>
```

Module Member (Module-based)

```
<module member>     When a module member is selected the operation
                    creates or updates the appropriate snapshot
                    branch. Module members are only a valid arguments
                    when using module snapshots.  You must specify the
                    module member as a workspace objects, for
                    example:
                    ./[<folder_name>]/<module_member>
```

```
                    <Module_Instance_Name>/[<folder_name>]/<module_member>
```

Workspace Module (Module-based)

```
 <workspace module>   When a workspace module instance is specfied,
                      the operation tags the specified module version
                      in the vault.

                      Note: When -recursive is used with a workspace
                      module, the tag operation is still server-based
                      and will follow the hierarchy for the selected
                      module version on the server, not the one in the
                      workspace, if they are different.
```

External Module (Module-based)

```
 <external module>   Specifies the module instance of the external
                     module in the workspace or the URL of the
                     external module version to which you wish to
                     create the connection.  An external module is an
                     object or set of objects managed by a different
                     code management system but available for viewing
                     and integration through DesignSync.
                     Specify the external module in the workspace as a
                      module instance name.
                     Specify the external module Server URL as follows:
             sync[s]:///ExternalModule/<external-type>/<external-data>
                    where ExternalModule is a constant that identifies
                    this URL as an external module URL,
                    <external-type> is the name of the external module
                    procedure, and <external-data> contains the
                    parameters and options to pass to the
                    procedure. These parameters and options can be
                    passed from the procedure to the external code
                    management system or to DesignSync.

                    Note: In order to specify an external module, you
                   must have previously populated the module in the
                   workspace. You may also specify the external
                   module by href name only.
```

DesignSync Object (File-based)

```
 <DesignSync object> Tags the vault corresponding to the specified
                     local managed object.
```

DesignSync Folder (File-based)

```
<DesignSync folder> Tags the vaults corresponding to the objects
                    in the specified folder. If the folder contains
                    a legacy module, tag handles the module as a
                    customary DesignSync workspace, tagging the
                    objects and continuing to traverse the hierarchy
                    according to the -recursive option selected.
                    Note, however, that the tag command will not stop
                    at submodule boundaries.  (The -nomodulerecursive
                    option is no longer supported.)
```

OPTIONS

-branch

```
-branch <branch>    Tags the branch specified by the branch or
| -branch           version tag, auto-branch selector, or branch
  auto(<branch>)    numeric. This option overrides the object's
                    persistent selector list. If <branch> resolves
                    to a version, the branch of that version is
                    tagged. The -version and -branch options are
                    mutually exclusive. The -branch option is not
                   applicable to operations on a module snapshot.

                    For a tag using an auto-branch selector, for
                    example Auto(Golden), if 'Golden' exists as a
```

branch, the 'Golden:Latest' version is tagged. If no branch named 'Golden' exists for the object, the tag operation fails.

Note: The -branch option accepts a single branch tag, a single version tag, a single auto-branch selector tag, or a branch numeric. It does not accept a selector or selector list.

-[no]comment (Module-based)

-[no]comment "<text>"	Specifies whether to tag the specified objects with a description attached. By default (-comment), tag requires a comment. Comments that exceed 1024 characters are truncated to the first 1024 characters. Enclose the description in double quotes if it contains whitespace. When you tag the objects, DesignSync appends tag comments, if there are any, to existing comments to create a "version log". The ampersand (&) and equal (=) characters are replaced by the underscore (_) character in revision control notes. Note: If -comment is specified with -replace, the comment replaces the existing tag comment. If -nocomment is specified, the existing tag comment is removed.

-[no]delete (Module-based)

-[no]delete	Indicates whether to delete the specified version or branch tag. When specified with module members in a module snapshots, it removes the members from the snapshot. Note: Because a tag can apply to either a branch or a version (not both), DesignSync determines which kind of tag is specified and deletes it. You can define access controls to selectively control the deletion of branch and version tags. The -delete option is mutually exclusive with the -branch, -replace, and -version options. You cannot specify a specific version or branch because only one version or branch of an object can have a given tag, so just specifying the object itself is sufficient.

-[no]delete (File-based)

-[no]delete	Indicates whether to delete the specified version or branch tag. Note: Because a tag can apply to either a branch or a version (not both), DesignSync determines which kind of tag is specified and deletes it. You can define access controls to selectively control the deletion of branch and version tags. The -delete option is mutually exclusive with the -branch, -replace, and -version options. You cannot specify a specific version or branch because only one version or branch of an object can have a given tag, so just specifying the object itself is sufficient.

-exclude (Module-based)

-exclude <objects>	Specifies a comma-separated list of objects to exclude from the operation. Wildcards are allowed. Do not specify paths in your arguments to -exclude. Before operating on each object (such as during a recursive tag operation), DesignSync compares the object's leaf name (path stripped off) to the exclude list to see if there is a match. Because the object's path is removed, the object will not match any object in the exclude list specified with a path. For example, if you specify '-exclude bin/*.exe', you will not successfully exclude bin/foo.exe or any other *.exe file. You need to instead specify '-exclude *.exe', or '-exclude foo.exe' if you want to exclude only 'foo.exe'. The result is that you cannot exclude a specific instance of an object -- you exclude all matching objects. In addition to objects you specify using the -exclude option, the "These objects are always excluded" field from the Tools->Options-> General->Exclude Lists dialog box in the DesignSync graphical user interface lists objects that are always excluded from revision-control operations. Note: The -exclude option only applies to snapshot module tagging. It does not apply to

module objects tagging because you tag entire modules. For module objects, tag silently ignores the -exclude option.

-exclude (File-based)

-exclude <objects> Specifies a comma-separated list of objects to exclude from the operation. Wildcards are allowed.

Do not specify paths in your arguments to -exclude. Before operating on each object (such as during a recursive tag operation), DesignSync compares the object's leaf name (path stripped off) to the exclude list to see if there is a match. Because the object's path is removed, the object will not match any object in the exclude list specified with a path. For example, if you specify '-exclude bin/*.exe', you will not successfully exclude bin/foo.exe or any other *.exe file. You need to instead specify '-exclude *.exe', or '-exclude foo.exe' if you want to exclude only 'foo.exe'. The result is that you cannot exclude a specific instance of an object -- you exclude all matching objects.

In addition to objects you specify using the -exclude option, the "These objects are always excluded" field from the Tools->Options-> General->Exclude Lists dialog box in the DesignSync graphical user interface lists objects that are always excluded from revision-control operations.

-filter (Module-based)

-filter <objects> Specify one or more extended glob-style expressions to identify an exact subset of objects on which to operate.

The -filter option takes a list of expressions separated by commas, for example:
-filter +top*/.../*.v,-.../a*

Prepend a '-' character to a glob-style expression to identify objects to be excluded (the default). Prepend a '+' character to a glob-style expression to identify objects to be included. Note that if the list of expressions begins with an include character ('+'), the filter excludes all objects except those that match the include string.

Specify the paths in your glob-style expressions relative to the current directory, because DesignSync matches your expressions relative to that directory. For submodules followed through hrefs, DesignSync matches your expressions against the objects' natural paths, their full relative paths. For example, if a module Chip references a submodule, CPU, and CPU contains a file, '/libs/cpu/cdsinfo.tag', DesignSync matches against '/libs/cpu/cdsinfo.tag', rather than matching directly within the 'cpu' directory.

Note: Workspace module members are only tagged individually when working with snapshot branches.

If your design contains symbolic links that are under revision control, DesignSync matches against the source path of the link rather than the dereferenced path. For example, if a symbolic link exists from 'tmp.txt' to 'tmp2.txt', DesignSync matches against 'tmp.txt'. Similarly for hierarchical operations, DesignSync matches against the unresolved path. If, for example, a symbolic link exists from dirA to dirB, and dirB contains 'tmp.txt', DesignSync matches against 'dirA/tmp.txt'.

The extended glob-style expressions you use to filter the objects are standard glob-style expressions, but they are extended so that you can use the "..." syntax to indicate that the expression matches any number of directory levels. For example, the expression, "top/.../lib/*.v" matches *.v files in a directory path that begin with "top", followed by zero or more levels, with one of those levels containing a lib directory. The command traverses the directory structure. If a directory name matches an exclude clause of the filter, then the entire directory and all its contents are filtered (the command stops descending at that point), otherwise the command continues traversing the directory structure searching for matching objects.

The -filter option does not override the exclude list set using either the -exclude option or SyncAdmin's General=>Exclude Lists tab; the items in the exclude list are combined with the filter expression. For example, an exclude list of "*%,*.reg" combined with '-filter .../*.doc' is equivalent to: '-filter

.../*.doc,.../*%,.../*.reg'.

-[no]modified (File-based)

-[no]modified	Indicates whether to tag the versions in the vault corresponding to the modified objects in your work area. If you specify -nomodified (default), when the tag operation encounters a locally modified object, the operation displays an error for the object and does not tag any version of that object in the vault. Note: This option affects modified objects only. If a work area object is unmodified, the tag operation tags the version in the vault that matches the one in your work area.

-modulecontext (Module-based)

-modulecontext <context>	Specifies the workspace module context to include in a module snapshot. This allows you to restrict the tag operation to a specified module. This option is only applicable to module snapshots.

-[im]mutable (Module-based)

-[im]mutable	Indicates whether a module's generated tag is to be immutable (fixed) or mutable. Use an immutable tag if the state of the module you are tagging is to be retained indefinitely. Use a mutable tag (default) if you want to reapply the tag to a newer snapshot of the module. The -mutable and -immutable tags apply only to modules. The options are ignored for other objects being tagged including module snapshots. By default, you cannot move or delete an immutable tag; however, you can override this behavior by applying the -immutable tag with the -replace or -delete option. You can move or delete a mutable tag using the -replace or -delete option without having to specify a mutability option. To convert an immutable tag to a mutable tag, first delete the immutable tag (using tag with the -immutable and -delete options). Then, create a mutable tag.

As a team leader, you might want to prevent members from moving or deleting immutable tags even if they apply the -immutable option. To do so, see ENOVIA Synchronicity Access Control Guide: "Access Controls for Tagging".

-[no]recursive (Module-based)

-[no]recursive — Specifies whether to perform this operation on the specified folder or module, or to traverse its subfolders and hierarchy. The -recursive option, when used on a module, traverses the hierarchical references in static mode on the server. For more information on tagging modules recursively, see Using Tags on Module Versions. The default value is -norecursive.

If you specify a local folder (a folder in your work area), the tag operation uses the local folder hierarchy to determine which objects to tag in the vault.

By default, the tag operation is nonrecursive; it tags the specified module, or members in the current directory only.

-[no]recursive (Legacy-based)

-[no]recursive — Specifies whether to perform this operation on the specified folder or to traverse its subfolders. The default value is -norecursive.

If you invoke 'tag -recursive' and specify a local folder that is the base directory of a legacy module configuration, the command tags all objects in the folder and its subfolders. Then the command follows the configuration's hierarchical references (hrefs) and tags all objects in referenced submodule configurations. The -nomodulerecursive option is no longer supported; thus, it is not possible to prevent tag from following legacy module hrefs.

If you specify a vault folder for a recursive tag operation, the operation does not follow the legacy module hrefs.

If a legacy module configuration has hrefs to submodules whose base directories reside outside the directory hierarchy where the tag operation started, objects in those submodules are not

tagged.

The recursive tag operation handles a configuration-mapped vault folder on the SyncServer based on the method you used to populate the DesignSync configuration to your work area. See "Interaction with Legacy Modules" for information.

By default, the tag operation is nonrecursive; it tags objects in the specified folder only.

-[no]recursive (File-based)

-[no]recursive — Specifies whether to perform this operation on the specified folder or to traverse its subfolders. The default value is -norecursive.

If you specify a local folder (a folder in your work area), the tag operation uses the local folder hierarchy to determine which objects to tag in the vault. If you specify a vault folder, the operation traverses the vault folder hierarchy, tagging objects in that hierarchy.

When 'tag -recursive' is invoked, the DesignSync client scans all of the subdirectories, determining the file versions to tag. Tag requests are then sent to the server, with up to 1000 objects included in each request. On the server, each file version in the request is processed. For each file version, access control is checked, and the file version then tagged (if allowed by access controls). The results from processing the set of up to 1000 objects are then returned to the DesignSync client. The next set of up to 1000 objects are then processed by the server, and so on.

By default, the tag operation is nonrecursive; it tags objects in the specified folder only.

-[no]replace (Module-based)

-[no]replace — Indicates whether to move the tag to the target version or branch, even if the specified tag is already in use on another version or branch. By default (-noreplace), 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 that you can move a tag

from a branch to a version or a version to a branch. DesignSync provides a warning message when you do so.

Notes:
o When -replace is used on a module snapshot, it replaces the tag on the specified module members in the snapshot.

o If you specify a comment, the tag operation replaces the comment with the new comment. If you do not specify a comment, the operation removes the previous comment associated with tag.

-[no]replace (File-based)

-[no]replace

Indicates whether to move the tag to the target version or branch, even if the specified tag is already in use on another version or branch. By default (-noreplace), 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 that you can move a tag from a branch to a version or a version to a branch. DesignSync provides a warning message when you do so.

Note: If you specify a comment, the tag operation replaces the comment with the new comment. If you do not specify a comment, the operation removes the previous comment associated with tag.

-report

-report <mode>

Specifies the contents of a report on the tag operation.

Available modes are:
o brief - This mode lists:
- Objects that were not tagged.
- Objects skipped by the tag operation because it created a new configuration map.
- A count of successes and failures for the tag operation. Note: This count is output only if you are using the stcl/stclc command shell.

If the -report option is not specified, the default mode is '-report brief'.

o normal - This mode provides the same output as the brief mode but in addition lists objects that were successfully tagged. (Default)

o verbose - Displays the same information as 'normal' and a skip notice for any objects excluded by the -filter or -exclude options.

-[no]selected

-[no]selected — Indicates if the command should use the select list (see the 'select' command) or only the arguments specified on the command line.

-noselected indicates that the command should not use the select list. (Default) If -noselected is specified, but there are no arguments selected, the tag command fails, even if there are valid arguments in the select list.

-selected indicates that the command should use the select list and any objects specified on the command line. If -selected is not specified, and there are no objects specified on the command line, the tag command uses the select list for the command.

-trigarg

-trigarg <arg> — Specifies an argument to be passed from the command line to the triggers set on the tag operation. If the argument contains whitespace, enclose the argument within double quotation marks ("") if using the dss command shell or braces ({}) if using the stcl command shell.

-version (Module-based)

-version <selector> — Specifies the version to tag. If the selector resolves to a branch, the Latest version on that branch is tagged. By default (-version not specified), the current version in your work area is tagged. The -version and -branch options are mutually exclusive. The -version option is not applicable to operations on a module snapshot.

```
                    If you specify a date selector (Latest or
                    Date(<date>)), DesignSync augments the
                    selector with the persistent selector list
                    to determine the version to be tagged. For
                    example, if the persistent selector list
                    is 'Gold:,Trunk' and you specify
                    'tag -version Latest <tag>', then the
                    selector list used for the tagging
                    operation is 'Gold:Latest,Trunk:Latest'.

                    Note:

                    To use -version to specify a branch, specify
                    both the branch and version as follows:
                    '<branchtag>:<versiontag>', for example,
                    'Rel2:Latest'. You can also use the shortcut,
                    '<branchtag>:', for example "Rel2:".  If
                    you don't explicitly specify the branch
                    selector in this way, DesignSync does not
                    resolve the selector as a branch selector.
```

-version (File-based)

```
 -version <selector> Specifies the version to tag. If the selector
                    resolves to a branch, the Latest version on
                    that branch is tagged. By default (-version not
                    specified), the current version in your work
                    area is tagged. The -version and -branch
                    options are mutually exclusive.

                    The -version option checks for retired files (as
                    of version 4.2 sp1). If the selector resolves to a
                    branch that is retired, it skips tagging the retired
                    files.

                    Note: The retired state only affects adding or
                    replacing a version tag when -version is specified.
                    It does not affect deleting a version tag.

                    If you specify a date selector (Latest or
                    Date(<date>)), DesignSync augments the
                    selector with the persistent selector list
                    to determine the version to be tagged. For
                    example, if the persistent selector list
                    is 'Gold:,Trunk' and you specify
                    'tag -version Latest <tag>', then the
                    selector list used for the tagging
                    operation is 'Gold:Latest,Trunk:Latest'.

                    Note:

                    To use -version to specify a branch, specify
                    both the branch and version as follows:
                    '<branchtag>:<versiontag>', for example,
```

'Rel2:Latest'. You can also use the shortcut, '<branchtag>:', for example "Rel2:". If you don't explicitly specify the branch selector in this way, DesignSync does not resolve the selector as a branch selector.

-warn

-warn <mode>

Provides additional checks depending on the <mode>. The -warn option supports the 'exists' mode, which makes sure the named object still exists before allowing the tag. This is rarely needed and only applicable in those cases where someone else has removed the vault file since you checked it out. This could happen if:

- A UNIX 'rm' command was used. (Note: 'rm' is not recommended; use 'rmvault' instead.)
- The 'rmvault -nokeepvid' command was used, then the object was checked in again with 'ci -new'. (Note: The '-nokeepvid' option is not recommended; use the default option, '-keepvid'.

-xtras (Module-based)

-xtras <list>

List of command line options to pass to the external module change management system. Any options specified with the -xtras option are sent verbatim, with no processing by the populate command, to the Tcl script that defines the external module change management system.

--

--

Indicates that the command should stop looking for command options. Use this option when an argument to the command begins with a hyphen (-).

RETURN VALUE

In dss/dssc mode, you cannot operate on return values, so the return value is irrelevant.

```
In stcl/stclc mode, two lists are returned. The first list is a
count of objects successfully processed; the second list is a count
of objects that failed to be processed. The first list is non-empty
if at least one object was successfully processed. The second
list is non-empty if at least one object failed.

Notes:
 - Scripts should only test for non-empty lists to
   determine success or failure. The actual content of a non-empty
   list currently takes the form "Objects succeeded (n)" and
   "Objects failed (n)", where 'n' is the number of objects that
   succeeded or failed. However, these return values are subject to
   change in a future release.
 - If all objects fail, an exception is thrown.
```

SEE ALSO

```
ci, co, command defaults, mkbranch, populate, select, setselector,
url tags, url resolvetag, vhistory
```

EXAMPLES

Example of Tagging a Module with an Immutable Tag (Module-based)

```
This example tags the Latest version of a module with an -immutable
(fixed) tag.  To tag modules, you specify the server URL of the
module:

  stcl> tag -immutable GOLD sync://guaraldi:30089/Modules/Chip
```

Example of Tagging All Files Matching a Wildcarded String (File-based)

```
This example adds the REL3_STABLE version tag to all managed '.v'
files in the current work area except template.v. Because -version
and -branch are not specified, the versions corresponding to the
current (last-fetched) objects in the work area are tagged.
```

```
stcl> tag -exclude template.v REL3_STABLE *.v
```

Example of Tagging a Specified Version of Files (File-based)

This example tags the 'Latest' versions on the current branch of all '.v' and '.h' files in the current work area with the 'stable' tag, moving the tag from older versions if necessary.

```
stcl> tag -version Latest -replace stable *.v *.h
```

Example Showing Tagging Modified File in the Workspace (File-based)

This example shows that you cannot tag a file version you have modified in your work area. (To tag the modified file version, you must first check it in and then use 'tag' to tag the new version in the vault.) To tag the version in the vault instead of the modified version in your work area, you can use the -modified option:

For this example, assume that the file alu.v has been modified.

```
dss> tag test alu.v

Beginning Tag operation...

Tagging: sync://alusrvr.ABCo.com:30090/Projects/ALU/alu.v;1.1 :
Failed: Modified object exists in the workspace.
To tag the modified version, check it in and then tag it.
To tag the version in the vault, use tag -modified.

Tag operation finished.

{} {Objects failed (1)}

dss> tag test alu.v -modified

Beginning Tag operation...
Tagging: alu.v   : Added tag 'test' to version '1.1'
Tag operation finished.

{Objects succeeded (1)} {}
```

Example of Tagging Locked Files (File-based)

This example shows that tagging a locked file tags the last-fetched version (in this example, version 1.1). You cannot tag the upcoming version (1.2).

```
dss> co -lock -nocom test.asm
Checking out: test.asm : Success - Checked Out version: 1.1 -> 1.2
```

```
dss> tag Alpha test.asm

Beginning Tag operation...
Tagging: test.asm   : Added tag 'Alpha' to version '1.1'
Tag operation finished.

{Objects succeeded (1)} {}

dss> tag -version 1.2 -replace Alpha test.asm
Tagging: sync:///test.asm;1.2:Failed:som:
Error 88: Tag:Version doesn't exist
Tag operation finished.

{} {Objects failed (1)}
```

Example of Using Exclude to Restrict Which Files are Tagged (File-based)

o This example shows the exclude syntax for vault objects.

```
stcl> ls [url vault .]

Directory of: sync://srv2.ABCo.com:2647/Projects/Sportster/code

Time Stamp             WS Status  Version   Type        Name
----------             ---------  -------   ----        ----
12/28/2005 11:00                                      samp.asm;
12/28/2005 11:00                                      samp.lst;
12/28/2005 11:00                                      samp.mem;
12/28/2005 11:00                                      samp.s19;
12/28/2005 11:00                                      sample1.asm;
12/28/2005 11:00                                      test.asm;
12/28/2005 11:00                                      test.mem;
stcl> tag -exclude {samp.asm;1,test.mem;1} -rec -version Latest \
stcl> testtag [url vault .]

Beginning Tag operation...

samp.asm;1 : Excluded from operation by filter
Tagging:
sync://srv2.ABCo.com:2647/Projects/Sportster/code/sample1.asm;1
  : Added tag 'testtag' to version '1.2'
test.mem;1 : Excluded from operation by filter
Tagging:
sync://srv2.ABCo.com:2647/Projects/Sportster/code/samp.s19;1
  : Added tag 'testtag' to version '1.1'
Tagging:
sync://srv2.ABCo.com:2647/Projects/Sportster/code/samp.mem;1
  : Added tag 'testtag' to version '1.2'
Tagging:
sync://srv2.ABCo.com:2647/Projects/Sportster/code/test.asm;1
  : Added tag 'testtag' to version '1.2'
Tagging:
sync://srv2.ABCo.com:2647/Projects/Sportster/code/samp.lst;1
  : Added tag 'testtag' to version '1.1'
```

```
Tag operation finished.

{Objects succeeded (5)} {}
stcl>
```

Example of Tagging a Branch (File-based)

```
This example adds the branch tag 'Rel2.1' to the Trunk branch of all
files in a work area (a recursive tag):

  dss> tag -recursive -branch Trunk Rel2.1 .
```

Example of Deleting a Tag (File-based)

```
This example deletes a version tag 'gold' and a branch tag
'Rel2.1'. Note that the syntax is the same; DesignSync determines if
the specified tag is a branch tag or a version tag.

  dss> tag -delete gold samp.lst
  Deleting Tag: samp.lst         : Deleted version tag 'gold'
  dss> tag -delete Rel2.1 samp.lst
  Deleting Tag: samp.lst         : Deleted branch tag 'Rel2.1'
```

Example of Adding a Tag to a Branch or Version (File-based)

```
o This example adds a 'Gold' branch tag to the branch tagged
  Silver, or if no such branch exists, the branch associated with
  the version tagged Silver:

  dss> tag -branch Silver Gold test.asm
```

Examples of Tagging an Object on the Server (File-based)

```
The following two examples add a tag 'beta' to the 1.2 version of
'top.v'. If 'top.v' is in the local work area, you would specify
'top.v' as the argument with a '-version 1.2' option. But in this
case, the version object itself is specified, so 'top.v' need not be
in your work area.  The first example uses version-extended naming.
The second example uses the -version option.

  dss> tag beta sync://apollo:2647/Projects/Sportster/code/top.v;1.2

  dss> tag beta -version 1.2 \
  sync://apollo:2647/Projects/Sportster/code/top.v
```

Advanced Revision Control

import

import Command

NAME

```
import              - Fetches an object, leaving it unmanaged
```

DESCRIPTION

```
This command fetches local copies of the specified objects from the
specified vault to your current workspace. Unlike fetching with the
"co" command, imported files do not retain their association with the
vault (are no longer managed).

The "import" command can be used to switch an object's vault
association. Perform the import on the object and then run the ci
command on the  new, unmanaged, object to check it into the new
vault.

Note: The selector list can be used to select what versions to fetch.
If the select list is used, it is inherited from parent folder (the
folder into which the objects are imported). If the selector is not
appropriate for the vault from which you are importing use the
-version option to specify the version. For DesignSync objects, the
selector list will pick up tagged versions or version numbers.  For
modules, the selector list can only specify version numbers.
```

SYNOPSIS

```
import  [-force] [-version <selector>] [--]
       <argument> <object> [<object>...]
```

ARGUMENTS

- Module URL (Module-based)
- Vault URL (File-based)

Module URL (Module-based)

```
<module URL>        Specifies the DesignSync URL of the module for the
                    object being imported.  Specify the URL (for
                    example:
                    sync://srvr2.ABCo.com/Modules/Chip/chip.c; )
                    when the object being imported is a member of a
                    module.
```

Vault URL (File-based)

<vault URL>	Specifies the DesignSync vault URL for the object being imported. Specify the vault (for example: sync://system:30138/Projects/Sportster/test/runit;) when the object being imported is not a member of a module.

OBJECTS

- Module Member (Module-based)
- DesignSync File Object (File-based)

Module Member (Module-based)

<module member>	Specifies the module member to import. You cannot import folders.

DesignSync File Object (File-based)

<DesignSync object>	Specifies the file object to import. You cannot import folders.

OPTIONS

- -force
- -version (Module-based)
- -version (Legacy-based)
- -version (File-based)
- --

-force

-force	Overwrites a local object if the object has the same name as an object being imported. When -force is not specified, the default behavior is to not overwrite local objects and return an error message explaining why the objects were not imported.

-version (Module-based)

```
-version <selector>  Specifies the version of the objects being
                     imported.

                     If no version is specified, the default version
                     imported is the latest object version in the
                     module version specified by the module URL
                     argument.

                     Note: To use -version to specify a branch,
                     specify both the branch and version as follows:
                     '<branchtag>:<versiontag>', for example,
                     'Rel2:Latest'. You can also use the shortcut,
                     '<branchtag>:', for example "Rel2:".  If
                     you don't explicitly specify the branch
                     selector in this way, DesignSync does not
                     resolve the selector as a branch selector.
```

-version (Legacy-based)

```
-version <selector>  Specifies the version of the objects or
                     individual member vault being imported.

                     If no version is specified, DesignSync inherits
                     the selector of the parent folder (the folder
                     into which the objects are imported).

                     Note: To use -version to specify a branch,
                     specify both the branch and version as follows:
                     '<branchtag>:<versiontag>', for example,
                     'Rel2:Latest'. You can also use the shortcut,
                     '<branchtag>:', for example "Rel2:".  If
                     you don't explicitly specify the branch
                     selector in this way, DesignSync does not
                     resolve the selector as a branch selector
```

-version (File-based)

```
-version <selector>  Specifies the version of the objects or
                     individual member vault being imported.

                     If no version is specified, DesignSync inherits
                     the selector of the parent folder (the folder
                     into which the objects are imported).

                     Note: To use -version to specify a branch,
                     specify both the branch and version as follows:
                     '<branchtag>:<versiontag>', for example,
                     'Rel2:Latest'. You can also use the shortcut,
                     '<branchtag>:', for example "Rel2:".  If
                     you don't explicitly specify the branch
                     selector in this way, DesignSync does not
                     resolve the selector as a branch selector.
```

--

```
--                  Indicates that the command should stop looking for
                    command options. Use this option when arguments
                    to the command begin with a hyphen (-).
```

RETURN VALUE

```
none
```

SEE ALSO

```
co, populate, selectors
```

EXAMPLES

- Example of Importing a Specific Module Version (Module-based)
- Example of Importing a Module Member (Module-based)
- Example of Moving Files to a New Vault Associated with a Workspace (File-based)

Example of Importing a Specific Module Version (Module-based)

```
This example fetches a specific version of a module object by its
natural path.

  dss> import sync://cassini:2647/Modules/Chip;1.5 /libs/df2test/cdsinfo.tag
```

Example of Importing a Module Member (Module-based)

```
This example shows fetching a specific module member vault version
using the -version option to specify the version number.

  dss> import -version 1.3 sync://h:p/Modules/Chip;1.5\
  /libs/df2test/cdsinfo.tag
```

Example of Moving Files to a New Vault Associated with a Workspace (File-based)

```
This example performs a "switch vault" operation, where files from
one vault are imported into a work area, then checked into another
vault (the vault associated with the work area).

  dss> scd /users/jane/myworkdir
```

```
  dss> import -version Trunk sync://cassini:2647/Projects/Saturn/Rocket \
       rover.doc lander.doc
  rover.doc:   Success Imported
  lander.doc:  Success Imported

  dss> ls rover.doc lander.doc
  Time Stamp         Status  Version          Locked By   Name
  ----------         ------  -------          ---------   ----
  05/04/2000 09:24        -  Unmanaged                    rover.doc
  05/04/2000 09:24        -  Unmanaged                    lander.doc
```

Jane can now check these files into the vault associated with her work area:

```
  dss> ci -new -nocom -keep rover.doc lander.doc
```

mkbranch

mkbranch Command

NAME

```
mkbranch            - Creates a new branch
```

DESCRIPTION

- Branching Modules (Module-based)
- Branching File-based Objects (File-based)

```
This command creates a new branch for the specified objects. The new
branch is tagged with the specified branch name (sometimes called a
branch name "tag".  For more information, see the tag help
topic). The branch-point version -- the version off which the branch
is created -- depends on the object type:

The 'mkbranch' command does not set the local workspace to use the
new branch (your local metadata is not modified). If you want future
operations to take place on the new branch, change your persistent
selector to point to the appropriate branch. For example:
    dss> mkbranch Dev top.v
    dss> setselector Dev:Latest top.v

In addition to the manual creation of branches with 'mkbranch',
which supports the "project branching" design methodology,
DesignSync supports the "auto-branching" design methodology.
See the "selectors" help topic for more information.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

This command supports the command defaults system.
```

Branching Modules (Module-based)

```
For a module, the branch point is the module version specified in the
command.
    Notes:
    * The branch point version is created as the first module version
      on the new branch.

    * To verify the mkbranch on the module, you can use the contents
      commands to see the module manifest. If you use the vhistory
      command with the -report +Q option, you see the module objects
      in an added state, but you do not see the hierarchical
      references.
```

Branching File-based Objects (File-based)

```
The behavior of the file-based object branch depends whether you
branch from the workspace object version or the server object
version.

  o For a local object (file or collection object, or local folder
    if you specify -recursive), the branch point is the
    last-retrieved (current) version in your work area. If
    DesignSync cannot determine the current version (for example,
    the object is not under revision control), the mkbranch command
    fails. If the local object is locked (for example, you have
    version 1.4 -> 1.5), the branch is still created off the
    current version (1.4), because the upcoming version (1.5) does
    not yet exist in the vault. You cannot specify the -version
    option with local objects.

  o For a vault, or vault folder if you specify -recursive, you
    must specify the -version option and provide a selector list
    (typically a version tag or version number) to identify the
    branch-point version.
```

SYNOPSIS

```
mkbranch [-[no]comment <text>] [-exclude <string>,[<string>...]]
         [-[no]recursive] [-[no]selected] [-version <selector>] [--]
         <branchname> [<argument> [<argument> ...]]
```

ARGUMENTS

- Branch Name (Module-based)
- Branch Name (Legacy-based)
- Branch Name (File-based)

- Server Module Version (Module-based)
- DesignSync Object (File-based)

Branch Name (Module-based)

<branchname>	Specifies the name to use for the new branch. Note: DesignSync vaults and legacy module branch names cannot end in --R.

Branch Name (Legacy-based)

<branchname>	Specifies the name to use for the new branch. Note: Branch names cannot end in --R.

Branch Name (File-based)

<branchname>	Specifies the name to use for the new branch. Note: Branch names cannot end in --R.

Server Module Version (Module-based)

<Server Module version>	Specifies the server module version to branch. To reduce the possibility of inadvertently branching the wrong module version, you must specify the version number either with this argument or by using the -version option. Note: You always branch on the server, not in the workspace

DesignSync Object (File-based)

<DesignSync object>	Specifies the DesignSync object or folder to branch.

OPTIONS

- -[no]comment (Module-based)
- -exclude
- -[no]recursive (File / Legacy-based)
- -[no]selected
- -version

- --

-[no]comment (Module-based)

```
-[no]comment <text>   Specifies the comment to include with the newly
                      created module branch.

                      -nocomment stores no comment to explain the
                      purpose of the branch.(Default)
                      -comment <text> stores the value of <string>
                      as the branch comment. To specify a multi-word
                      comment, enclose the text string in quotation
                      marks (""). The comment is attached to the
                      branch itself and to the branch tag.

                      Note: If there is a minimum comment length
                      defined with SyncAdmin for the client, you must
                      specify a comment for mkbranch.  This option
                      does not check the Access Control comment length
                      for the checkin command.
```

-exclude

```
-exclude <objects>    Specifies a comma-separated list of objects
                      to be excluded from the operation. (Legacy
                      modules only) Wildcards are allowed.

                      Do not specify paths in your arguments to
                      -exclude. Before operating on each object
                      (such as during a recursive 'mkbranch'),
                      DesignSync compares the object's leaf name
                      (path stripped off) to the exclude list to see
                      if there is a match. Because the object's path
                      is removed, the object will not match any
                      object in the exclude list specified with a
                      path. For example, if you specify '-exclude
                      bin/*.exe', you will not successfully exclude
                      bin/foo.exe or any other *.exe file. You need
                      to instead specify '-exclude *.exe', or
                      '-exclude foo.exe' if you want to exclude only
                      'foo.exe'. The result is that you
                      cannot exclude a specific instance of an
                      object -- you exclude all matching objects.

                      In addition to objects you specify using the
                      -exclude option, the "These objects are always
                      excluded" field from the Tools->Options->
                      General->Exclude Lists dialog box in
                      DesignSync lists objects that are always
                      excluded from revision-control operations.
```

-[no]recursive (File / Legacy-based)

-[no]recursive	Determines whether to branch the specified object or any subfolders. -norecursive branches only the specified object.(Default) -recursive branches the specified object and any subfolders of the object. If the command argument is a vault folder, DesignSync operates on all vaults in the folder and its subfolders. If the command argument is a local folder, DesignSync operates on all files and collections objects in the folder and its subfolders. If a recursive 'mkbranch' operation encounters a vault folder on the SyncServer that is configuration-mapped to another vault folder (using DesignSync REFERENCEs), the 'mkbranch' operation creates a new configuration map on the SyncServer instead of branching the objects in the vault folder. Note: You can recursively branch vaults within legacy module folders.

-[no]selected

-[no]selected	Indicates if the command should use the select list (see the 'select' command) or only the arguments specified on the command line. -noselected indicates that the command should not use the select list. (Default) If -noselected is specified, but there are no arguments selected, the mkbranch command fails, even if there are valid arguments in the select list. -selected indicates that the command should use the select list and any objects specified on the command line. If -selected is not specified, and there are no objects specified on the command line, the mkbranch command uses the select list for the command.

-version

-version <selector>	Specifies the version off of which the branch is created. This option is required unless the argument contains a version specifier.

```
                    Notes:
                    o You can specify a dynamic selector as the
                      argument to the -version option, for example,
                      '-version Rel2:Latest'; however, doing so is
                      not recommended because you are attempting
                      to freeze dynamically changing objects.
                      Instead, specify a fixed version selector,
                      for example, '-version rel2_revision1'.
                    o If you do choose to specify a branch using
                      the -version option, specify both the branch
                      and version as follows:
                      '<branchtag>:<versiontag>', for example,
                      'Rel2:Latest'. You can also use the shortcut,
                      '<branchtag>:', for example "Rel2:".  If
                      you don't explicitly specify the branch
                      selector in this way, DesignSync does not
                      resolve the selector as a branch selector.
```

--

```
  --                Indicates that the command should stop looking for
                    command options. Use this option when arguments
                    to the command begin with a hyphen (-).
```

RETURN VALUE

```
In stcl/stclc mode, two lists are returned, where the first list
is non-empty if at least one object was successfully processed,
and the second list is non-empty if at least one object failed.

Notes:
 - Scripts should only test for non-empty lists to
   determine success or failure. The actual content of a non-empty
   list currently takes the form "Objects succeeded (n)" and
   "Objects failed (n)", where "n" is the number of objects that
   succeeded or failed. However, these return values are subject to
   change in a future release.
 - If all objects fail, an exception occurs (the return value is
   thrown, not returned).
```

SEE ALSO

```
selectors, setselector, select, tag, command defaults
```

EXAMPLES

```
Note: The following examples demonstrate the syntax and behavior of
mkbranch, but do not necessarily show a recommended use model.
```

Example Showing Module Branching (Module-based)

```
In the following example, a new "Dev" branch is created off the
current version of the SPC module. Note that the persistent
selector has not changed after you run the mkbranch command.  In
order to work on the new branch, you should manually change the
persistent select list.

  dss> mkbranch Dev sync://srvr2.ABCo.com/Modules/SPC;1.4

  Beginning MkBranch operation...

  Branching:   sync://srvr2.ABCo.com:2647/Modules/SPC;1.4 : Success -
  Created branch 1.4.1, tagged Dev
  MkBranch operation finished.

  {Objects succeeded (1)} {}
```

Example of Branching Two Files From Your Workspace (File-based)

```
In the following example, a new "Dev" branch is created off the
current versions of two local files. Note that the persistent
selector list before and after the operation is unchanged; you must
manually change the persistent selector list if you want to work on
the new branch. Note: This example demonstrates the syntax and
behavior of 'mkbranch', but does not show a recommended use
model. When branching individual objects (as opposed to entire
projects), use the auto-branching option on checkin or checkout.

  dss> ls -report PR samp.asm test.mem
  Selector    Version          Name
  --------    -------          ----
  Trunk       1.3              samp.asm
  Rel2.1      1.2.1.2          test.mem

  dss> mkbranch Dev samp.asm test.mem

  Beginning MkBranch operation...

  Branching: samp.asm   : Success - Created branch 1.3.1, tagged Dev
```

```
  Branching: test.mem   : Success - Created branch 1.2.1.2.1, tagged Dev

  MkBranch operation finished.

  dss> ls -report PR samp.asm test.mem     <== Selector is unchanged
  Selector    Version          Name
  --------    -------          ----
  Trunk       1.3              samp.asm
  Rel2.1      1.2.1.2          test.mem

  dss> setselector Dev samp.asm test.mem   <== Set the selector to
                                           <== work on the new branch

  Beginning Set Selector operation...

  Finished Set Selector operation.

  dss> ls -report PR samp.asm test.mem
  Selector    Version          Name
  --------    -------          ----
  Dev         1.3              samp.asm
  Dev         1.2.1.2          test.mem
```

Example Showing Branching The File Objects in the Workspace Recursively (File-based)

The following example branches an entire work area. The -recursive option traverses the hierarchy and creates a branch called "Rel2.1" off the current version of every object:

```
  dss> mkbranch -recursive Rel2.1 .
```

Example Showing Branching the Server Version of a File (File-based)

The following example makes a branch called "main" off version 1.1 on a file "test.mem". Because the vault for "test.mem" is specified, the -version option is required to identify the branch-point version. Note that there must have already been two branches off version 1.1, because the new branch is 1.1.3.

```
  stcl> mkbranch main -version 1.1 [url vault test.mem]

  Beginning MkBranch operation...

  Branching:   sync://myhost:2647/Projects/Sportster/code/test.mem; :
  Success - Created branch 1.1.3, tagged main

  MkBranch operation finished.

  {Objects succeeded (1)} {}
```

In the previous example, if version 1.1 of "test.mem" had a version tag called "Alpha", you could specify that tag instead of the

```
version number:

  stcl> mkbranch main -version Alpha [url vault test.mem]
```

Example Branching the Entire Project from the Server (File-based)

```
The following example is typical of a release engineer creating a
new project branch off an existing configuration, for example
to create a patch release:
  stcl> mkbranch -version Rel3.1 -rec Patch1 sync://host:2647/Projects/Asic

The following example also shows a release engineer branching an
entire project. In this case, the engineer has a local work area
populated with the latest versions of all files. He places
version tags on these latest versions to identify the branch points
for the new branch. He then branches from those branch-point versions.

  stcl> populate -recursive
  stcl> tag -recursive bp-Rel30 .
  stcl> mkbranch -recursive -version pb-Rel30 [url vault .] Rel30
```

mkfolder

mkfolder Command

NAME

```
mkfolder            - Creates a folder (directory)
```

DESCRIPTION

```
This command creates one or more folders (directories), either on
the local file system or on the server.

o You can specify the folder as a relative path, an absolute path,
  a "file:" URL, or a "sync:" URL.
o The permissions of the new folder are inherited from the parent
  folder.
o When creating local folders (not specifying the "sync:" protocol),
  you must have write privileges for the parent directory.
o When specifying a folder name that contains whitespace, use
  double quotes.
o DesignSync creates whatever folders are needed to create
  the specified path (similar to UNIX's 'mkdir -p' command).
o The ability to create server-side folders ("sync:" protocol) can
  be accessed controlled using the MakeFolder action.
o When creating folders, you must use a legal name.  If characters
  are restricted, you cannot use them in folder names.  For more
  information on restricted characters, see Exclude Lists in the
```

```
   DesignSync Data Manager Administrator's Guide.
This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.
```

SYNOPSIS

```
mkfolder [--] <foldername> [<foldername>...]
```

OPTIONS

- --

--

```
  --          Indicates that the command should stop looking for
              command options. Use this option when arguments
              to the command begin with a hyphen (-).
```

SEE ALSO

```
mvfolder, rmfolder
```

EXAMPLES

```
The following examples show variations of creating folders:
 dss> mkfolder asic                        # relative path
 dss> mkfolder ../asic                     # relative path
 dss> mkfolder /home/goss/Projects/asic    # absolute path
 dss> mkfolder file:////home/goss/Projects/asic   # file: protocol
 dss> mkfolder sync://holzt:2647/Projects/asic   # sync: protocol
 dss> mkfolder asic1 ../asic2              # create two folders
 dss> mkfolder "asic 1"                    # foldername has whitespace
 dss> mkfolder asic/decoder/synth          # creates asic and decoder
                                           #   folders if necessary
```

mvfile

mvfile Command

NAME

```
mvfile              - Moves the specified object
```

DESCRIPTION

```
This command moves or renames the specified object. The object can be
a file or a Cadence cell view (collection object).

Notes: To move a Cadence collection, use mvfile with the
       -noallconfigs option.  No other collection types can be
       moved with mvfile.

With the mvfile command, you can specify a relative or absolute
path to the object. You can move unmanaged and managed objects. To
move a folder, use the mvfolder command.

For unmanaged objects, the mvfile operation is equivalent to an
operating-system move operation (for example, Unix 'mv').

For managed objects, the mvfile operation when not using -allconfigs:
  1. Moves the object locally, if run from the workspace.
  2. Retires the current branch of the object's vault.
  3. Creates a new vault for the new object name. The initial
     version in the new vault is created from the current object
     in your work area (as though you executed 'ci -new' on the
     object after it was moved locally). At this time, a default
     check-in comment is added. The comment includes the user,
     time of check-in, and version URL from where the new vault
     originated.
By leaving the existing vault intact, configurations containing the
object prior to being moved are preserved. The new vault contains
only version 1.1 of the moved object; the new vault does not
contain the version history of the moved object. The branch tags
of the current branch, if any and the selector at the time of the
move are retained for the moved object. The fetch state of the
moved object is the same as the original object; for example,
if you had a link to a cached file prior to the move, you
have a link to a cached file after the move.

For managed objects, the mvfile operation with -allconfigs option:
  1. Moves the object locally, if done from the workspace.
  2. Creates a new vault and then moves all configurations of
     the original object to the new object.
  3. Removes the old vault.
     The new vault contains all versions that were there in the
     old vault. So, if the old vault had file X with version Y,
     the new vault also has file X with version Y. It also contains
     the version history of the moved object. In addition, since
     the old vault is removed, all existing configurations will
     contain the new vault name and not the original vault name.

Note: The mvfile command with or without the -allconfigs option
      does not update the "Objects" list on RC notes.

When using the -noallconfigs option (default), you cannot move:

- Server-side objects (as specified with the sync:// protocol).
- DesignSync references. You must fetch a local copy, link to the
```

```
  cache, or link to the mirror first.
- Generic collection objects.
- Objects on a locked branch. You must release the lock first.
- Managed symbolic links to files or folders.
- Collection members, such as the files that make up a Cadence
  cell view.

When using the -allconfigs option, you cannot move:

- Cache or mirror links. You must fetch a local copy or a reference first.
- Collection objects.
- Objects on a locked branch. You must release the lock first.
- Managed symbolic links to files or folders.
- Collection members, such as the files that make up a Cadence
  cell view.

The following additional restrictions apply:

- The source and destination locations must be on the same file
    system.
- The destination object or vault cannot already exist.
- The path of the destination object must already exist; no
  components of the destination path are created for you.
- When not using -allconfigs option with managed objects, the
  checkin to create the new vault is always done with the
  '-keys kkv' option; revision-control keywords, if any, are
  retained and their values are updated.
- For managed objects, the vault folders for both source and
  destination locations must be on the same SyncServer
  (the vault set to the same sync://<host>:<port>).
  Otherwise, an error about crossing domain boundaries results.
- When not using the -allconfigs option with managed objects, you
  cannot rename the object, and then rename it back to the original name.
- Cadence cell view (.sync.cds) collection objects can only be moved to a
  Cadence cell folder.
- Cadence cell view objects must always be named with a .sync.cds
  extension.
- ProjectSync notes do not migrate to the new vault, unless the
  -allconfigs option is used.
- When the -allconfigs option is used, RevisionControl notes are not
  affected, remaining associated with the original vault.
- If there are restricted characters are enabled, no restricted
    character can be used in the natural path of the object.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

This command supports the command defaults system.
```

SYNOPSIS

```
mvfile [-allconfigs] <object> <destination>
```

ARGUMENTS

Object

```
object          The object that you want to move or rename. The
                object can be a local file or Cadence cell view
                (.sync.cds) collection object. You can specify an
                absolute or relative path.
```

Server URL object

```
serverURL       Specifies the URL of object to be moved. Specify the
                URL as follows:
                sync://<host>[:<port>]/<path>/<filename>; or
                syncs://<host>[:<port>]/<path>/<filename>;
                where 'sync://' or 'syncs://'
                are required, <host> is the machine on which the
                SyncServer is installed, and <port> is the SyncServer
                port number (defaults to 2647/2679).
                For example:
  sync://serv1.abco.com:1024/Projects/Chip/Chip.c;

                If you specify a server URL file, you must use the
                -allconfigs options to move the entire vault.
```

Destination

```
destination     The destination can be:
                  - The folder into which the object is
                    moved. The object retains its current name.
                  - The new name of the object (a rename operation).
                  - Both a folder and new name for the moved
                    object.
                You can specify an absolute or relative path.
```

Server URL Destination

```
serverURL       Specifies the URL of new object location. Specify the
                URL as follows:
                sync://<host>[:<port>]/<path>/<filename>; or
                syncs://<host>[:<port>]/<path>/<filename>;
                where 'sync://' or 'syncs://'
                are required, <host> is the machine on which the
```

```
                SyncServer is installed, and <port> is the SyncServer
                port number (defaults to 2647/2679).
                For example:
  sync://serv1.abco.com:1024/Projects/NewChip/NewChip.c;
```

OPTIONS

- -[no]allconfigs

-[no]allconfigs

```
 - [no]allconfigs   Determines whether to update the location of the
                    specified objects in all configurations that use
                    the object.

                    -noallconfigs does not update the configurations.
                    When this option is specified, ProjectSync notes
                    linking to the old objects are not updated and
                    might break.

                    -allconfigs moves the vault file in the repository
                    and, if successful, moves file in the local
                    workspace transferring the metadata from the old
                    file to the new file. This updates all
                    configurations containing objects to the new
                    object. All attachments, such as ProjectSync
                    notes, are migrated to the new vault of the moved
                    object.

                    When attempting mvfile -allconfigs where the vault
                    is on a pre-4.2 server, you will get an error
                    message.
```

RETURN VALUE

```
None. An exception is thrown if the command fails.
```

SEE ALSO

```
mvfolder, rmfile, retire, ci, command defaults
```

EXAMPLES

- Example Showing Renaming a File
- Example Showing Moving the File to a New Folder
- Example Showing Renaming a File on the Server

- Example Showing Moving a File on the Server
- Example Showing Renaming and Moving a Cadence Cell View
- Example Showing The History of a Moved File

Example Showing Renaming a File

```
The following example renames a file:
  dss> mvfile top.v mod1.v
  top.v: Success Moved
```

Example Showing Moving the File to a New Folder

```
The following example moves a file to a new folder (newdesign)
while keeping the same name (test.mem):
  dss> mvfile ../test.mem /home/tgoss/newdesign
  test.mem: Success Moved

Note that in the preceding example, if the object is managed,
the destination folder must previously have had its vault set
to the same SyncServer as the original folder or an error will
result.
```

Example Showing Renaming a File on the Server

```
The following example renames a file on the server.
 dss> mvfile -allconfigs
sync://srv1.ABCo.com:2647/Projects/ChipDesign/Chip.c;
sync://srv1.ABCo.com:2647/Projects/ChipDesign/NewChipDesign.c;

 Chip.c;: Success Moved
```

Example Showing Moving a File on the Server

```
This following example moves a file to a different folder on the
server.
dss> mvfile -allconfigs
sync://srv1.ABCo.com:2647/Projects/ChipDesign/Chip.c;
sync://srv1.ABCo.com:2647/Projects/NewChipDesign/Chip.c;

 Chip.c;: Success Moved
```

Example Showing Renaming and Moving a Cadence Cell View

```
The following example moves a Cadence cell view to a new folder and
renames it. Note that a cell view collection object must have a
.sync.cds extension:
```

```
dss> mvfile symbol.sync.cds ../and5/symbol2.sync.cds
symbol.sync.cds: Success Moved
```

Example Showing The History of a Moved File

```
The following example demonstrates that the move of a managed
object retires the current branch of the existing vault, and creates
a new vault with no version history. The renamed object retains the
original object's branch tags and selector. The 'vhistory' command
is used before and after the move for comparison purposes.

  stcl> vhistory top.v
  Object:          file:///home/tgoss/Projects/Sportster/top/top.v
  Vault URL:       sync://myserver:30020/Projects/Sportster/top/top.v;
  Current version: 1.2.1.1
  Current state:   Copy
  ----------------------------
  Branch:       1.2.1
  Branch tags:  rel21

  ----------------------------
  Version:      1.2.1.1
  Version tags: Latest
  Derived from: 1.2
  Date:         Tue Feb 22 14:05:46 2000; Author: tgoss
  Branching off for rel21 release

  ================================================================

  stcl> url selector top.v
  rel21
  stcl> mvfile top.v mod1.v
  top.v: Success Moved
  stcl> vhistory mod1.v
  Object:          file:///home/tgoss/Projects/Sportster/top/mod1.v
  Vault URL:       sync://myserver:30020/Projects/Sportster/top/mod1.v;
  Current version: 1.1
  Current state:   Copy
  ----------------------------
  Branch:       1
  Branch tags:  rel21

  ----------------------------
  Version:      1.1
  Version tags: Latest
  Date:         Tue Feb 22 14:06:24 2000; Author: tgoss

  ================================================================

  stcl> url selector mod1.v
  rel21
  stcl> ls top.v
  No such object: file:///home/tgoss/Projects/Sportster/top/top.v
  stcl> co -get -version rel21 top.v
```

```
  Beginning Check out operation...

  Checking out: top.v           : Success - Fetched  version: 1.2.1.1

  Checkout operation finished.

  {Objects succeeded (1)} {}
  stcl> ls -nohead top.v
  Copy       <Retired> Up-to-date  1.2.1.1         top.v
  stcl>
```

mvfolder

mvfolder Command

NAME

```
mvfolder            - Moves the specified folder
```

DESCRIPTION

```
This command moves or renames the specified folder. This folder may
exist in either the workspace, on the server, or in both places. You
can specify a relative or absolute path. To move a file or Cadence
cell view, use mvfile.

You can move a managed or unmanaged folder. For unmanaged folders,
the mvfolder operation is equivalent to an operating-system move
operation (for example, Unix 'mv'). A managed folder is one that has
a corresponding vault folder (a setvault has been performed on the
the folder). Both the local folder and vault folder are moved.
You can use the 'mvfolder' command only on a vault folder on the
server, without affecting your workspace.  There is an example of
this in the Examples section.

IMPORTANT:
  If you use mvfolder to rearrange the subdirectory structure within a
  vault folder hierarchy, the hierarchy of the original structure is lost.
  Because directories are not versioned, whatever data is fetched will be
  fetched into local workspace directories whose structure mimics that of
  the vault, at that point in time. Use mvfolder on a managed folder only
  if you are certain about changing the vault's directory structure.

You cannot move:
  - Your current folder or any parent folder.
  - Cadence cell-view folders. A cell-view folder is moved as part
    of a mvfile of a Cadence cell view (.sync.cds) collection object.

The following additional restrictions apply:
  - The source and destination locations must be on the same file
    system.
```

```
  - For managed folders, the vault folders for both source and
    destination folders must be on the same SyncServer
    (the vault set to the same sync://<host>:<port>).
    Otherwise, an error about crossing domain boundaries results.
  - The path of the destination folder must already exist; no
    components of the destination path are created for you.
  - A Cadence cell folder can only be moved to a Cadence library
    folder.
  - The folder names must contain only printable characters, and
    may not include any characters that have been restricted.  For
    more information on restricted characters, see Exclude Lists in
    the DesignSync Administrator's Guide.

Note: If the folder is a legacy module configuration, notify project
leaders and module administrators to remove each hierarchical
reference specifying the old project folder and add a new
hierarchical reference for the new project folder. For more
information, see the edithrefs, addhref and rmhref commands.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

Any attachments, such as ProjectSync notes, associated with objects
in a folder are retained. Notelinks are updated to reflect
the new vault path.

This command supports the command defaults system.
```

SYNOPSIS

```
mvfolder [--] <folder> <destination>
```

ARGUMENTS

- Folder
- Destination

Folder

```
folder          The local or server-side folder that you want to move
                or rename. You can specify an absolute or relative
                path.
```

Destination

```
destination     The destination can be:
                  - The folder into which the folder is moved.
                    The source folder retains its current name.
                  - The new name of the folder (a rename operation).
```

```
  - Both a folder and new name for the
    moved folder.

The path to the destination must exist; no portions
of the path are created for you. You can specify an
absolute or relative path.

If the destination already exists as a folder, then
the folder being moved is placed inside of the
destination folder.
```

OPTIONS

- --

--

```
 --             Indicates that the command should stop looking for
                command options. Use this option when arguments
                to the command begin with a hyphen (-).
```

RETURN VALUE

```
None. An exception is thrown if the command fails.
```

SEE ALSO

```
mvfile, rmfolder, mkfolder, scd, command defaults
```

EXAMPLES

- Example Showing Renaming a Folder
- Example Showing Moving a Folder to a New Path
- Example Showing Moving a Cadence Cell to a Different Library
- Example Showing a Vault Rename

Example Showing Renaming a Folder

```
The following example renames a folder:
  dss> mvfolder mod1 mod2
  mod1: Success Moved
```

Example Showing Moving a Folder to a New Path

```
The following example moves a folder to a new folder (newdesign)
while keeping the same name (mod1):
  dss> mvfolder ../mod1 /home/tgoss/newdesign
  mod1: Success Moved

Note that in the preceding example, if a setvault has been applied
to the original folder, the destination folder must previously have
had its vault set to the same SyncServer or an error will result.
```

Example Showing Moving a Cadence Cell to a Different Library

```
The following example moves a Cadence cell to a different library
and renames it.
  dss> mvfolder ASIC/and2 TTLLib/and
  and2: Success Moved
```

Example Showing a Vault Rename

```
The following example shows a vault folder being renamed directly,
without requiring a corresponding workspace directory.

stcl> scd sync://lotti:30158/Projects
stcl> ls

Directory of: sync://lotti:30158/Projects

Time Stamp             WS Status  Version   Type        Name
----------             ---------  -------   ----        ----
                                                       Test
stcl> mvfolder Test TestHere
Test: Success Moved
stcl> ls

Directory of: sync://lotti:30158/Projects

Time Stamp             WS Status  Version   Type        Name
----------             ---------  -------   ----        ----
                                                       TestHere
stcl>
```

purge

purge Command

NAME

```
purge               - Purges specified branches or versions of
                      objects from the vault
```

DESCRIPTION


```
This command deletes specified branches or versions of
an object on a single branch in the vault. You can use this command
to clean up the vault by deleting old versions of objects. You also
can remove an entire branch: all branch tags, all version data, and
all version tags on the deleted branch.

Your server must be at release DS 4.2, or higher, to use this command
to delete branches. Your server must be at release V6R2012 to use
this command to purge module branches or versions.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

This command supports the command defaults system.
```

Restrictions

```
o The purge command does not follow DesignSync REFERENCEs or
  hierarchical references and operate on referenced data. If you
  use 'purge -recursive' on a directory hierarchy, the purge
  operation skips hierarchical elements based on REFERENCEs or
  hierarchical references.

o The purpose of the purge command is to purge old versions on a
  single branch in the vault. The command does not support
  selectors with multiple entries. If you specify a purge of a
  vault object, you specify the branch to purge and the purge
  operation deletes versions from that branch only. If you specify
  a purge of a workspace object but you do not specify the -branch
  option, the purge operation uses the current branchid of the
  object and purges only that branch. The operation ignores a
  selector.

o You can specify a purge of objects in a workspace or directly
  defined by their vault URLs. If you specify a purge of a
  workspace directory, the purge operation traverses the workspace
  hierarchy to find all the objects for the purge operation. Then
  the operation deletes corresponding vault objects that have the
  appropriate branchid. If you use the purge command on workspace
  objects, it is possible that the purge operation will remove a
  version in the vault that is currently located in your
  workspace. This action is deliberate, since 'purge' is intended
  to clean up the vault independent of existing workspaces.
```

Note: The purge command deletes versions from the vault; the command never affects data in your workspace.

- The 'purge -recursive' command fails if you specify a directory for which you have not set the vault, even if that directory contains a subdirectory for which the vault is set. For example, suppose you have the following directory hierarchy in your workspace:
  Directory A (vault not set)
    Directory B (vault not set)
      Directory C (vault is set)

  If you use 'purge -recursive' and specify Directory A or Directory B, the purge operation fails and never finds Directory C. In this example, to purge files in Directory C, you must specify that directory with the 'purge -recursive' command.

- To remove a branch with side branches, you must first remove the side branches.

- Locked branches cannot be removed. You must first unlock the branch before deleting it.

- The branch numbers of deleted branches cannot be reused.

Triggers and Revision Control Notes and 'purge'

The purge command generates the same triggers and creates the same revision control notes as the rmversion command. Administrators should carefully consider which pre- and post-command triggers and revision control notes to enable. For example, the purge command causes triggers for 'rmversion'.

Error Handling

If an error occurs, the purge command reports the error but does not throw an exception. The command proceeds with all other objects still left in the list of objects to purge.

The purge operation reports an error only when:
- There is an error in the command option
- Objects specified do not exist
- None of the versions selected for the purge can be deleted

Failure to delete a version or a failure to access an object is reported in the overall failure count for the operation, not as an error.

Using Purge with Modules (Module-based)

```
Module versions and module members can be purged.

Module members are not purged explicitly, but are purged when all
module versions referencing the module member version are purged.

Module instances cannot be specified with a wildcard, such as '*'.
Branch tags must be specified by the branch name or numeric.

You cannot delete:
 - Branchpoint versions (for example, if 1.2.1 is a branch, you
 cannot delete version 1.2)
 - Version 1.1
 - The only version on a branch if it is not a .1 version
 - The Latest version of a locked branch
 - Tagged versions (unless the -force switch is used).
 - Version .1 when:
    o Another version on the branch could not be deleted.
    o Additional branch tags exist on the branch specified with
    the -branch switch and you do not specify '-force.'
    o Additional branch tags exist on the branch, the -branch switch
    is used with a branch numeric, and you do not specify '-force.'
    o The object is a module.  The initial module version on any
      branch can not be purged, even when specified with the
      '-force' option, unless the entire branch is purged.
 - Module member versions which are not explicitly purged.
```

Using Purge with Files-Based Objects (File-based)

```
With this command you can specify:
 - One or more objects in your workspace.
 - A folder in your workspace. (Note: You must specify '-recursive'
   in order to purge a folder.)
 - A vault URL for the object or folder.
 - A branch to remove.

Note: Wildcards (such as '*') are allowed for all arguments except
module instance names.

You cannot delete:
 - Branchpoint versions (for example, if 1.2.1 is a branch, you
 cannot delete version 1.2)
 - Version 1.1
 - The only version on a branch if it is not a .1 version
 - The Latest version of a locked branch
 - Tagged versions (unless the -force switch is used).
 - Version .1 when:
    o Another version on the branch could not be deleted.
    o Additional branch tags exist on the branch specified with
    the -branch switch and you do not specify '-force.'
    o Additional branch tags exist on the branch, the -branch switch
    is used with a branch numeric, and you do not specify '-force.'
 - The Latest version on branch 1 if the vault was previously
 removed with 'rmvault -keepvid' and then recreated.
```

```
Note:  Do not attempt to remove an entire vault by specifying branch
  1. Instead, use the rmvault command to delete the entire vault.
```

SYNOPSIS

```
purge [-branch <branchname>] [-[no]dryrun]
      [-exclude <object>[,<object>...]] [-[no]force]
      [-keepsince <date>] [-keepversions <n>][-[no]recursive]
      [-report <mode>] [--] <argument> [<argument>...]
```

ARGUMENTS

- Module URL (Module-based)
- Module Workspace (Module-based)
- DesignSync Object (File-based)
- DesignSync Folder (File-based)

Specify one or more of the following arguments:

Module URL (Module-based)

<module URL>	URL of the module containing the versions being purged. Specify the URL as follows: sync[s]://<host>[:<port>]/Modules/[<Category>...] /<module>; where <host> is the SyncServer on which the module resides, <port> is the SyncServer port number, <Category> identifies the optional category path, and <module> is the name of the module. Note: If you specify a module URL as the argument, you must specify the branch with the -branch option.

Module Workspace (Module-based)

<workspace module>	Specifies the module instance name or path of the module containing the versions being purged. By default, this will purge from the branch specified by the persistent selector on the workspace. Note: The purge command accepts version-extended workspace folder and file paths. It does not accept version-extended module instance names.

```
                    Also module instance names cannot be specified by
                    using wildcard characters.
```

DesignSync Object (File-based)

```
<DesignSync object> Purges the specified DesignSync object from the
                    server. This object can be a vault URL, or an
                    object in your workspace.
```

DesignSync Folder (File-based)

```
<DesignSync folder> Purges a local folder and the objects within it.
                    (Note: You must specify '-recursive' in order to
                    purge a folder.)
```

OPTIONS

- -branch
- -dryrun
- -exclude (File-based)
- -[no]force (Module-based)
- -[no]force (File-based)
- -keepsince
- -keepversions (Module-based)
- -keepversions (File-based)
- -recursive (File-based)
- -report
- --

-branch

```
-branch             Specifies the branch identifier of the objects you
<branchname>        want to purge. You can specify only one branch
                    with this option.

                    For the <branchname>, specify a branch tag (for
                    example, rel40) or branch numeric (for
                    example, 1.4.2) only. Do not specify a colon
                    (:) with the branchname.

                    If you do not specify the -branch option, the
                    purge command uses the current branch
                    identifier of the object.

                    Note: If you use a server URL to specify an object
                    for purging, you must also specify the -branch
                    option.
```

-dryrun

```
-[no]dryrun    Determines whether this command performs a purge, or
               creates a report showing what files will be purged when
               the command is run.
               The -nodryrun option performs the purge.(Default)
               The -dryrun option generates the list of what will be
               purged when the purge command is run.
               This option can be used with any of the report modes.
```

-exclude (File-based)

```
-exclude <objects>
                  Excludes one or more objects (files,
                  collections, or folders) from the purge
                  operation. Specify objects in a comma-separated
                  list. Wildcards are allowed.

                  The purge operation excludes objects specified
                  with '-exclude' in addition to objects always
                  excluded from revision-control operations.
```

-[no]force (Module-based)

```
-[no]force     Determines whether this command purges a tagged
               version or branch.
               -noforce ignores versions or branches that are
               tagged. (Default)
               -force purges the tagged versions and branches along
              with the rest of the specified versions. On vault
              objects, when used with -keepversions 0,' -force
              removes the branch (and the .1 version) even when the
              branch includes tags.

               Note: The branch point version cannot be removed until
                     all branches rooted to it are removed and the
                     first and latest versions on a branch cannot be
                     removed until the entire branch is removed.
```

-[no]force (File-based)

```
-[no]force     Determines whether this command purges a tagged
               version or branch.
               -noforce ignores versions or branches that are
               tagged. (Default)
               -force purges the tagged versions and branches along
              with the rest of the specified versions. On vault
```

```
objects, when used with -keepversions 0,' -force
removes the branch (and the .1 version) even when the
branch includes tags.
```

-keepsince

```
-keepsince <date>
                    Keeps all versions created after the specified
                    date and deletes the other versions of an
                    object on a single branch in the vault. For
                    example, purge -keepsince "10 September 2003"
                    keeps all versions of the object that were
                    created after 10 September 2003 and deletes all
                    other versions on the branch.

                    For <date>, specify a date or a
                    relative time, enclosed in double quotation
                    marks (" "). For example:
                    stcl> purge -keepsince "10 days ago" top.v

                    Note: If you use the -keepversions option in
                    combination with the -keepsince option, the
                    purge operation keeps those versions specified
                    by each option. For example, if you specify:

                    purge -keepversions 3 -keepsince "Jan 1 2004"

                    the purge operation deletes all versions of the
                    object from the vault except for the last 3
                    versions or any versions created after the Jan
                    1, 2004.

                    If you use the -keepsince option with
                    -keepversions 0 and -keepsince specifies that some
                    versions cannot be deleted, the branch is not
                    removed. The -keepsince option takes precedence
                    over -keepversions 0.
```

-keepversions (Module-based)

```
-keepversions <n>   Keeps the last <n> versions of an object (on a
                    single branch in the vault) and deletes the
                    other versions. The default is 1.

                    For example, 'purge -keepversions 3' keeps only
                    the last 3 versions of the object and deletes
                    all other versions from the vault.

                    To delete all versions, use '-keepversions 0'.

                    To delete a branch completely, identify the
                    branch with the -branch switch and specify
```

'-keepversions 0'. If you do not specify a branch, the current branch is picked up from the metadata when the command is issued on a workspace file.

Note: The initial version, 1.1 is always preserved, even if -keepversions 0 is applied to the Trunk branch of a module.

-keepversions (File-based)

-keepversions <n>	Keeps the last <n> versions of an object (on a single branch in the vault) and deletes the other versions. The default is 1. For example, 'purge -keepversions 3' keeps only the last 3 versions of the object and deletes all other versions from the vault. To delete all versions, use '-keepversions 0'. To delete a branch completely, identify the branch with the -branch switch and specify '-keepversions 0'. If you do not specify a branch, the current branch is picked up from the metadata when the command is issued on a workspace file.

-recursive (File-based)

-[no]recursive	Determines whether to perform the purge on objects in the specified folder or on objects in the specified folder and all subfolders in the hierarchy. The -norecursive option purges only objects in the specified folder. (Default) The -recursive options purges objects in the specified folder and all subfolders in the hierarchy.

-report

-report <mode>	Specifies the amount of output generated by the purge operation. Available modes are: o brief - Displays: - The name and version of each object successfully deleted. - A message if no versions are selected for deletion. - Error messages.

```
                    o normal (the default mode) - Displays:
                      - A statement that the command is gathering
                        versions to be deleted.
                      - A statement reporting the number of
                        versions being deleted.
                      - Versions successfully deleted, each with
                        its full vault URL.
                      - A message if no versions are selected for
                        deletion.
                      - Error messages.

                    o verbose - Displays:
                      - A statement that the command is gathering
                        versions to be deleted.
                      - Information on the progress of the command
                        as it gathers versions for deletion,
                        including:
                        - Each directory traversed
                        - Each object found
                        - The number of versions on the branch to
                          be deleted
                      - A list of versions deleted and versions
                        kept (with the reason why the version was kept)
                      - Summary information about versions gathered
                        for deletion.
                      - A statement reporting the number of
                        versions being deleted.
                      - Each item being skipped (with the reason it
                        is skipped)
                      - Versions successfully deleted, each with
                        its full vault URL.
                      - A message if no versions are selected for
                        deletion.
                      - Error messages.
```

--

```
 --             Indicates that the command should stop looking
                for command options. Use this option when
                arguments to the command begin with a
                hyphen (-).
```

RETURN VALUE

```
 The return value from the purge command is a string in the form:
     {Objects succeeded (n)} {Objects failed (n)}

 The Objects succeeded count is the number of versions that were
 successfully deleted.

 The Objects failed count includes objects for which the version
```

```
deletion failed, objects not under revision control, and objects
for which version information for the object could not be fetched
from the vault.
```

SEE ALSO

```
rmversion, rmvault, command defaults
```

EXAMPLES

Example of Purging All but a 4 Versions of a Collection Object

```
This example deletes from the vault all versions of the top_design
Milkyway collection object in the current workspace except for the
last 4 versions.

   stcl> purge -keepversions 4 top_design.sync.mw
```

Example of Using Keep Since to Maintain 30 Days of Versions

```
This example deletes from the vault all versions of each object in
the current workspace directory except for versions created in the
last 30 days. The purge operation works recursively through all of
the data in your workspace.

   stcl> purge -keepsince "30 days ago" -rec .
```

Example of Using both the -keepsince and -keepversions Options

```
This example deletes from the vault all versions of each object in
Ted's ALU workspace except for the last 5 versions of the object OR
versions that were created in the last 30 days. The purge operation
works recursively through all of the data in Ted's ALU workspace.

   stcl> purge -keepversions 5 \
               -keepsince "30 days ago" -rec /home/ted/ProjectWork/ALU
```

Example of Purging Versions from the Server

```
This example deletes all versions of each object in the asic folder
on the Trunk branch in the vault, except for the last 4 versions. The
purge operation deletes these objects from the vault whether they are
in the user's workspace or not.

stcl> purge -keepver 4 sync://S1.ABC.com:2647/Projects/asic -branch
     Trunk
```

Example of Making then Purging a Branch

```
This example shows the user creating a "V11" branch of the "runit"
file, then immediately removing that branch via 'purge'.  The
'vhistory' output preceding the 'purge' shows the branch and version
information, from the user's perspective.  The verbose output from
'purge' uses the internal representation of that same data.

stcl> mkbranch V11 runit

Beginning MkBranch operation...

Branching:   runit                          : Success - Created branch
1.1.2, tagged V11

MkBranch operation finished.

{Objects succeeded (1)} {}

stcl> vhistory -all runit
Object:     file:///c|/barbg/work dir/Sportster/test/runit
Vault URL:  sync://srv2.ABCo.com:2647/Projects/Sportster/test/runit;
Current version: Refers to: 1.1
Current state:   Reference
----------------------------
Branch:       1
Branch tags:  Trunk

----------------------------
Version:      1.1
Version tags: Latest
Date:         Fri Oct 28 16:13:52 2005; Author: tbarbg2

----------------------------
Branch:       1.1.2
Branch tags:  V11
This branch does not yet have any versions.

================================================================

stcl> purge -report verbose -branch V11 -keepversions 0 runit
Gathering versions for deletion...
    Object c:\barbg\work dir\Sportster\test\runit :
        0 existing version on branch "V11" (branchid "1.1.2") :
        This is a stub branch
```

```
        deleting version 1.1.2.1
Purge version gathering summary:
    Objects processed: 1
    Versions selected for removal: 1
    Versions retained: 0
Deleting 1 version...
sync://srv2.ABCo.com:2647/Projects/Sportster/test/runit;1.1.2.1: \
 Success Deleted
{Objects succeeded (1)} {}
stcl>
```

Example Showing Module Purge on the Trunk Branch (Module-based)

```
This example shows a purge on a module branch, Trunk.  Note that
both the initial version and the Latest version are not purged.

stcl> purge -branch Trunk -report verbose -keepversions 1 Chip%0
 Gathering versions for deletion...
    Object c:\Workspaces\Chip%0 :
        6 existing versions on branch "Trunk" (branchid "1") :
        keeping version 1.1 (required, "1.1" version)
        keeping version 1.2 (tag exists on module version)
        deleting version 1.3
        deleting version 1.4
        deleting version 1.5
        keeping version 1.6 (keepversions criteria)
 Purge version gathering summary:
    Objects processed: 1
    Versions selected for removal: 3
    Versions retained: 3
 Deleting 3 versions...

 sync://srv2.ABCo.com:2647/Modules/Chip: Removing module versions ...

  sync://srv2.ABCo.com:2647/Modules/Chip;1.3: Success deleted
  sync://srv2.ABCo.com:2647/Modules/Chip;1.4: Success deleted
  sync://srv2.ABCo.com:2647/Modules/Chip;1.5: Success deleted

  sync://srv2.ABCo.com:2647/Modules/Chip: Identifying member versions
  to remove ...

  .
  sync://srv2.ABCo.com:2647/Modules/Chip: Found 3 candidate member
  version to remove.

 sync://srv2.ABCo.com:2647/Modules/Chip: Removing member versions ...

 .
 sync://srv2.ABCo.com:2647/Modules/Chip: Removed 1 member versions.
 {Objects succeeded (1)} {Objects failed (1)}
```

retire

retire Command

NAME

```
retire              - Marks a branch as obsolete
```

DESCRIPTION

```
This command marks a branch of a given object as obsolete. Retiring
a branch:
  o Prevents the branch from participating in future 'populate with
    latest versions' operations
  o Prevents new versions from being created on the branch (unless
    the -new option is used during the checkin, in which case the
    branch is unretired)

When you perform a retire, the time, date, and the username of the
user performing the retire are recorded and can be viewed as part of
the branch's version history. If the file is unretired, the retire
information is removed and cannot be accessed again.

Tip: If you want to preserve the retire information in the version
history, you can include the information in the checkin comment
either by unretiring the file with the ci command with the -new
-comment options, or by doing a ci -force after performing a retire
-unretire on the file.

Note: If you specify a collection member as the object to be
operated on, DesignSync skips the object and warns that the
object is not versionable. If DesignSync attempts to operate on
a collection member specified implicitly (through the use of
wildcards or a recursive operation), DesignSync silently skips
the object. You can change this behavior by using the SyncAdmin
"Map operations on collection members to owner" setting. If you
select this setting and DesignSync attempts to operate on a
collection member during a revision control operation,
DesignSync determines the member's owner collection and
operates on the collection as a whole.

A typical first step when team members join a project is to populate
their work areas with the Latest versions from a given branch of all
active design objects. By retiring a branch, you prevent objects
that are no longer needed (are obsolete) from being fetched into their
work areas, thereby limiting exposure to the object.

You can perform many operations on retired branches. For example,
you can fetch data from a retired branch. You can also tag files
on a retired branch if they are in your workspace. For information
on when you cannot tag files on a retired branch, see the tag command's
-version option description. However, operating on an object on a
retired branch is not typical.

It is important to note that objects on retired branches remain
```

```
part of past configurations. When you use the populate command to
retrieve a particular configuration other than 'Latest', objects
from retired branches are fetched. The populate command determines
what configuration to retrieve from the persistent selector list of
the folder you are populating (Trunk:Latest by default, or as set
by the 'setselector' command). You can override the folder's
persistent selector list by specifying 'populate -version
<selectorList>'. The populate command fetches objects from retired
branches, thereby preserving past configurations, if the selector
used is any of the following:

  o A version tag other than 'Latest', even if the version tag
    points to the Latest version
  o A version number, even if that number corresponds to the
    Latest version
  o <branchtag>:Date(<date>) or <branchtag>:VaultDate(<date>)

Note: If the selector for a populate operation resolves to a
      branch, DesignSync augments the selector to be
      <branch>:Latest, meaning, 'Get the Latest version from
      the specified branch'. In this case, objects from
      retired branches are not fetched.

See the "selectors" help topic for more information on selectors
and the "populate" help topic for details on the populate command.

Caution:
  o You cannot retire the branch of an object that is locked unless
    you specify the -force option, which removes the lock even if it
    is held by someone else.
  o By default, the local copy of an object on a retired branch is
    deleted unless the local copy is modified or you have specified
    the -keep option. However, if you specify -force (to unlock the
    object first), even a modified local copy is deleted. To unlock
    the object but keep your local copy, specify both -force and
    -keep.

    Note that when you specify the -branch option, DesignSync does
    nothing to the local work area objects. The branch is retired,
    but the local object is never deleted and the object's local
    metadata is unchanged, even if you specified -force.

You can unretire a retired branch in two ways:
  o Execute a 'retire -unretire' command on the branch.
-OR-
  o Check in a new version of the object onto the retired branch
    using the '-new' option to the ci command.

To determine whether the branch is retired you can:

 o Use 'ls -report status' command and the List View in the graphical
   user interface indicate if the current branch of an object in your
   work area is retired.

 o Use the 'url retired' command or view the data sheet for the
   object's vault to see the retired status.
```

```
  o Use the vhistory command with the -BX options (or in report
    -normal or report -verbose mode) to see the state of the branch
    and the username, time and date associated with the retire.


This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

This command supports the command defaults system.
```

SYNOPSIS

```
retire [-branch <branch> | -branch auto(<branch>)] | [-[no]force]
       [-[no]keep] [-[no]recursive] [-[un]retire] [-[no]selected]
       [-trigarg <arg>] [--] [<argument> [<argument>...]]
```

ARGUMENTS

Server URL

```
<server object>      Specify a vault or vault-folder object on the
                     server to retire in the specified branch.
```

Workspace Object

```
<workspace object>   Specify a versionable object in the local
                     workspace to retire in the specified branch.

                     Note:  If a workspace object is locked when the
                    retire operation is performed, the retire is
                    performed on the branch of the objects in the
                    workspace instead of on the selector for the
                    workspace. This can happen if the user changes
                    the selector but does not repopulate the
                    workspace with the updated branch files.
```

DesignSync Folder

```
<DesignSync folder>  Specify a folder containing DesignSync objects
                     with the -recursive option to retire all the
                     objects, including sub-folders, within a folder.
```

```
Notes:

You can retire only the objects within a folder,
without retiring the sub-folder structure, by
specifying the folder name with the * wildcard.

Non-module directories are not explicitly
checked into DesignSync so you cannot remove a
folder itself, only the contents of a folder.
If a folder becomes empty as a result of a
retire, it remains in the vault allowing you to
unretire the contents if needed.
```

OPTIONS

-branch

```
-branch <branch>    Retires the branch specified by the branch or
| -branch           version tag, auto-branch selector, or branch
  auto(<branch>)    numeric. The -branch option accepts a single
                    branch tag, a single version tag, a single
                    auto-branch selector tag, or a branch numeric.
                    It does not accept a selector or selector list.

                    This option overrides the object's persistent
                    selector list. If <branch> resolves to a version,
                    the branch of that version is retired.

                    If you do not specify the -branch option,
                    the command uses the branch specified
                    by the workspace selector or the sync URL to
                    retire the specified objects.

                    Notes:

                    When you use the -branch option,
                    DesignSync operates only on the vault and does
                    nothing with local work area objects. The local
                    object is never deleted and the object's local
                    metadata is unchanged.

                    If the retire command is run on a workspace
                    containing locked files and the -branch option is
```

not specified and -force is not specified, the retire fails. If -force is specified, the retire succeeds but is run against the branch currently populated in the workspace regardless of the selector value.

-[no]force

-[no]force

Specifies whether the branch should be removed even if the branch, or objects in the branch are locked.

-noforce does not remove the branch if the branch, or objects in the branch are locked. (Default)

-force unlocks locked branches (even if locked by someone else) prior to retiring them. This option also deletes local objects if the object was modified locally. To keep your local objects, specify -keep.

The -force and -branch options are mutually exclusive, because local objects are never deleted or otherwise affected when you specify -branch.

-[no]keep

-[no]keep

Specifies whether to keep or delete local copies of objects after their branches are retired. -nokeep deletes local objects unless either the object has been modified locally, or the -branch option is specified. (Default)

Note: When you use the -force option, even locally modified objects are deleted unless you explicitly specify -keep.

-keep preserves all local objects after the branch is retired. The objects become unmanaged by DesignSync.

If a locked reference is retired with -keep (and -force, to unlock the object's branch), a DesignSync reference remains in the workspace

You cannot use the -keep option if the local state of the object being retired is a link to the mirror. Because the SyncServer removes retired objects from the mirror directory, there would be no way to link to the retired

object.

The -keep option is ignored when unretiring an object, because 'retire -unretire' never affects local objects.

-[no]recursive

-[no]recursive	Indicates whether the retire command operates on the specified argument or all subfolders in the argument's hierarchy. -norecursive operates only on the specified argument. (Default) If the argument is a folder specified with a wilcard (<folder>/*), the contents of any sub-folders are not retired, but the contents of the specified folder are retired. -recursive operates on all subfolders in the specified argument's hierarchy.

-[un]retire

-[un]retire	Indicates whether the retire command is intended to retire the branch or reinstate the branch. -retire is provided to support the command defaults system. (Default) It retires the branch. -unretire reinstates branches as active, permitting future populate operations to fetch the latest objects on the branches. Note: An alternative way to unretire a branch is to perform a 'ci -new' command on an object. Refer to the ci command for details.

-[no]selected

-[no]selected	Determines whether the operation is performed just on the objects specified at the command line or on objects specified at the command line and objects in the select list (see the 'select' command) -noselected adds only objects specified on the command line. (Default) -selected adds objects specified on the command and in the select list.

-trigarg

```
 -trigarg <arg>      Specifies an argument to be passed from the
                     command line to the triggers set on the
                     retire operation. If the argument contains
                     whitespace, enclose the argument within
                     double quotation marks ("") if using the dss
                     command shell or braces ({}) if using the stcl
                     command shell.
```

--

```
 --                  Indicates that the command should stop looking for
                     command options. Use this option when arguments
                     to the command begin with a hyphen (-).
```

RETURN VALUE

```
 none
```

SEE ALSO

```
 url retired, ls, co, select, selectors, command defaults
```

EXAMPLES

- Example of Retiring Files
- Example of Retiring a Branch
- Example of Retiring a Specific File on a Branch

Example of Retiring Files

```
 The following example retires the current branches of any '.v'
 files in the current work area that begin with either 'cpu' or
 'mem' and removes the local copy of the file if present. After
 retiring the branch, users will not get these objects when
 populating the Latest versions from that branch, and new versions
 of the objects cannot be created on the retired branch.
   dss> retire cpu*.v mem*.v

 However, if a version of 'cpu1.v' is tagged 'rel1', then the following
 populate command successfully fetches that version even though the
 branch is retired:
   dss> populate -version rel1
```

Example of Retiring a Branch

```
The following example retires the 'Main' branch of all files in the
current folder. All local files remain and their metadata is
unchanged after the retire operation even though -keep was not
specified because -branch was specified.
  dss> retire -branch Main *
```

Example of Retiring a Specific File on a Branch

```
The following example retires the 'Main' branch of 'top.v' by
specifying its URL:
  dss> retire \
  "sync://apollo:2647/Projects/Sportster/top/top.v;Main:Latest"
or, in stcl/stclc mode, you might specify:
  stcl> retire [url vault top.v]Main:Latest
```

rmfile

rmfile Command

NAME

```
rmfile              - Deletes the specified object
```

DESCRIPTION

- Notes for Module Objects (Module-based)

```
This command deletes the specified object from the local file
system. The object can be a file or a collection object. You can
specify a relative or absolute path to the object. You cannot
delete an object on the server ('sync:' protocol). Deleting an
object does not affect the vault or module for that object.

Notes:
o You cannot delete a member of a DesignSync collection
  object.

o If you use rmfile to delete a collection object that has obsolete
  local versions, the command deletes all of the files making up
  those obsolete local versions.

This command supports the command defaults system.
```

Notes for Module Objects (Module-based)

```
If you use rmfile to delete an object that is a member of a module,
```

the object is removed from the workspace but remains a member of the module version on the server. To permanently remove items from a module, use the remove command. If module members are removed from the workspace, DesignSync places a marker in the workspace metadata that forces a full populate the next time the workspace is populated. For more information on full and incremental populate, see the populate command help.

You cannot remove mcache links with rmfile. To remove a mcache link use rmmod or rmfolder.

SYNOPSIS

```
rmfile [-trigarg <arg>] [--] <object> [<object> [...]]
```

ARGUMENTS

- Object

Object

object	The object that you want to delete. The object can be a local file or collection object. You can specify an absolute or relative path.

OPTIONS

- -trigarg
- --

-trigarg

-trigarg <arg>	Specifies an argument to be passed from the command line to the triggers set on the delete file operation. If the argument contains whitespace, enclose the argument within double quotation marks ("") if using the dss command shell or braces ({}) if using the stcl command shell.

--

--	Indicates that the command should stop looking for command options. Use this option when arguments to the command begin with a hyphen (-).

RETURN VALUE

```
none
```

SEE ALSO

```
command defaults, mvfile, remove, retire, rmfolder, rmvault, rmversion
```

EXAMPLES

Example of Removing a Specific File in the Current Working Directory

```
Delete top.v, which is in the current working directory:
  dss> rmfile top.v
  top.v: Success Deleted
```

Example of Removing Two Files

```
Delete two files: one absolute, one relative:
  dss> rmfile /home/Projects/ASIC/top.v ../decoder.v
  top.v: Success Deleted
  decoder.v: Success Deleted
```

Example of Removing a File with a Leading "-"

```
Delete a file called '-myfile':
  dss> rmfile -- -myfile
  -myfile: Success Deleted
```

Example of Removing a Member of a Collection

```
Deleting a file that is a member of a collection object fails. You
must delete the collection object itself. The following example
shows deletion of a Cadence cell view collection:
  dss> scd /home/Projects/smallLib/and2/verilog
  dss> rmfile pc.db
```

```
pc.db: Deletion of this object is not supported
Operation failed.
dss> scd ..
dss> rmfile verilog.sync.cds
verilog.sync.cds: Success Deleted
```

rmfolder

rmfolder Command

NAME

```
rmfolder            - Deletes the specified folder
```

DESCRIPTION

- Notes for Modules (Module-based)

```
This command deletes the specified folder from the local or server
file system. You can specify a relative or absolute path for a local
folder. Use the 'sync:' protocol to specify a server-side folder.

When this command is used with the -norecursive option, you cannot
delete a folder unless it is empty:
  - For local (client) folders, the folder cannot contain files,
    links, or folders. A folder containing a Synchronicity .SYNC
    metadata folder (for example, the folder you are deleting
    contains DesignSync references) can be deleted.
  - For server folders, the folder cannot contain vaults or other
    folders. A folder containing a sync_project.txt file can be
    deleted.

If your vault is associated with a mirror, any folder removed from
the vault is also removed from the mirror.

You cannot delete your current folder or any parent folder to your
current folder.

You cannot delete any folder or file if you do not have UNIX
permissions.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

This command supports the command defaults system.
```

Notes for Modules (Module-based)

```
If the folders are module objects, mcache links, module cache
```

```
folders, or, if rmfolder is used with the -recursive option, contain
module objects, the folder is removed from the workspace but remains
a member of the module version on the server. To permanently remove
items from a module, use the remove command.  If you remove mcache
link using rmfolder, the source mcache directory remains.

If module members are removed from the workspace, DesignSync places a
marker in the workspace metadata that forces a full populate the next
time the workspace is populated. For more information on full and
incremental populate, see the populate command help.
```

SYNOPSIS

```
rmfolder [-[no]keepvid] <folder> [-[no]recursive] [-trigarg <arg>]
         [<folder> [...]]
```

ARGUMENTS

- Folder

Folder

```
folder              The local or server-side folder that you
                    want to delete. You can specify an absolute
                    or relative path.
```

OPTIONS

- -[no]keepvid
- -[no]recursive
- -trigarg

-[no]keepvid

```
-[no]keepvid        Determines whether the version number of the
                    Latest version in a deleted vault (due to
                    'rmfolder -recursive' on a server-side folder) is
                    remembered. This behavior is important if a
                    vault of the same name is later created.

                    -nokeepvid does not store the version
                              number. (Default)

                    -keepvid stores the version number.

                    See the rmvault command for more details.
```

-[no]recursive

-[no]recursive	Determines whether to remove the specified folder and all subfolders in the hierarchy beneath it. The -norecursive option deletes the folder only if it's empty. This command is similar to the UNIX -rmdir command. The -recursive option deletes the contents of the folders and all subfolders. For local (client) folders, deletes as many folders and files as UNIX permissions allow. If an object was checked out "-lock" in the workspace being recursively removed, the lock is silently canceled prior to the object's removal. For server folders, access-control permissions are checked recursively for all vaults contained in the folders to be deleted. If all the access control checks pass, then the command deletes as many folders and files as UNIX permissions allow. If any access-control permission fails, the entire deletion operation is canceled. CAUTION: 'rmfolder -recursive', when used on a server folder, will delete vaults contained in the folder hierarchy even if a vault is locked or has one or more tagged versions. This behavior is in contrast to 'rmvault', which requires the -force option to delete a vault that is locked or has tagged versions. The default is -norecursive.

-trigarg

-trigarg <arg>	Specifies an argument to be passed from the command line to the triggers set on the delete folder operation. If the argument contains whitespace, enclose the argument within double quotation marks ("") if using the dss command shell or braces ({}) if using the stcl command shell.

RETURN VALUE

none

SEE ALSO

```
mkfolder, rmfile, rmversion, rmvault, remove, command defaults
```

EXAMPLES

Example of Removing Folder without Recursive

```
The following example demonstrates the use of rmfolder without the
-recursive option. The folder 'alu' contains one file, alu.v, which
must be deleted before the alu folder can be deleted.
  dss> rmfolder alu
  alu: som: Error 54:  Folder Not Empty.
  dss> rmfile alu/alu.v
  alu.v: Success Deleted
  dss> rmfolder alu
  alu: Success Deleted
```

Example of Removing Folders Recursively

```
The following example demonstrates the use of rmfolder with the
-recursive option. The folder 'alu' contains one file, alu.v, which
must be deleted before the alu folder can be deleted.
  dss> rmfolder -recursive alu
  alu: Success Deleted
```

Example of Removing a Folder on the Server

```
This example deletes an empty folder on a server:
  dss> rmfolder sync://holzt:2647/Projects/Sportster/Temp
  Temp: Success Deleted
```

Example of Removing a Folder Containing References

```
This example shows that you can delete a folder containing
references:
  dss> ls -report O

  Directory of: file:///home/ tgoss/Projects/Sportster/top/alu

  Object Type      Name
```

```
  -----------      ----
  Referenced File  alu.gv
  Referenced File  alu.v
  Referenced File  mult8.gv
  Referenced File  mult8.v
  dss> scd ..
  dss> rmfolder alu
  alu: Success Deleted
```

rmvault

rmvault Command

NAME

```
rmvault             - Deletes the specified vault
```

DESCRIPTION

```
This command deletes the specified vault. This command does not
remove any corresponding files in your local work area. This command
does not remove vaults in modules.  To remove modules, use the rmmod
command.  To remove objects in modules, use the remove command.

Important:
  Deleting a vault removes all versions of a design object from the
  SyncServer and should therefore be used with caution.  It is
  recommended that you use the retire command to retire a branch
  that is no longer used instead of deleting the vault. Use rmvault
  only when you are certain the vault will never again be needed
  and you need to reclaim disk space (for example, at the end of a
  project.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

You must use the -force option to delete a vault that has a locked
branch or has one or more tagged versions.

You can use version-extended names to specify a vault by specifying
a file followed by a semicolon (but no version number). For example,
"top.v;" is the vault specification for top.v.

   Notes:
     - You cannot use wildcards (such as '*') when using
       version-extended names.
     - When in stcl/stclc mode, you must surround version-extended
       names (or any URL with a semicolon) with double quotes.

When you delete a vault, you have the option of deleting all the
vault metadata or retaining metadata about the last
version number used by the vault. These behaviors, which are
```

```
controlled by the -nokeepvid and -keepvid options and a registry
setting, are important if a vault of the same name will later be
created. See the description of the -[no]keepvid options for details.

This command supports the command defaults system.
```

SYNOPSIS

```
rmvault [-[no]force] [-[no]keepvid] [-trigarg <arg>] [--]
<vault> [<vault> [...]]
```

OPTIONS

- -[no]force
- -[no]keepvid
- -trigarg
- --

-[no]force

```
-[no]force    Determines whether you can delete a vault with a locked branch
              or tagged versions.
              -noforce prevents you from from deleting a vault that has
                       tagged versions or locked branches. (Default)

                -force allows you to delete a vault that has tagged
                     versions or locked branches. Use this option
                     with caution:
                - A tagged version may be a necessary part of
                  a configuration.
                - A locked branch typically indicates someone is
                  editing the design object and therefore the
                  design object is still active.
```

-[no]keepvid

```
-[no]keepvid  Determines whether information about the version ID
              of the Latest version in the vault is retained, which
              is important if a vault of the same name is later
              created.

              For example, assume you delete the vault for top.v
              which has 1.1 as the Latest version, and you or another
              user later creates another top.v file (ci -new top.v). If
              you deleted the vault with -keepvid, the first version in
              the newly created vault is 1.2. If you specified
              -nokeepvid, the first version is 1.1.
```

The -nokeepvid behavior can cause problems if there are versions of the original top.v vault in mirrors, or user's work areas. For example, if a user's work area contains the old 1.1 version, DesignSync will not fetch the new 1.1 version when the user performs a populate. This would true anytime the latest version number of the new vault was the same as the version number in the workspace of the old vault. Had the vault been deleted with the -keepvid option, the populate would succeed, fetching version 1.2 (the first version in the newly created vault). Therefore, you should use -keepvid if a vault of the same name might later be created. However, the retained vault metadata does use a small amount of disk space, so if you want to reclaim all disk space used by a vault, use -nokeepvid.

If you do not specify -keepvid or -nokeepvid, the default is -keepvid behavior. You can redefine the default behavior using SyncAdmin. See "Command Defaults" in SyncAdmin help for details.

-trigarg

-trigarg <arg> Specifies an argument to be passed from the command line to the triggers set on the delete vault operation. If the argument contains whitespace, enclose the argument within double quotation marks ("") if using the dss command shell or braces ({}) if using the stcl command shell.

--

-- Indicates that the command should stop looking for command options. Use this option when arguments to the command begin with a hyphen (-).

RETURN VALUE

none

SEE ALSO

rmfile, rmversion, rmfolder, retire, command defaults

EXAMPLES

Example of Removing the Vault for a Single File

```
Delete the vault for top.v, where top.v is in my current work area.
The double quotes are required in stcl/stclc mode.
  stcl> rmvault 'top.v;'
```

Example of Removing the Vault using URL Vault

```
Delete the vault for top.v using the 'url vault' command instead of
version-extended naming:
  stcl> rmvault [url vault top.v]
```

Example of Removing the Vault using the Server URL

```
Delete all the vaults in the vault folder 'top'. This command
does not delete vaults in any folder under 'top' (rmvault
is not recursive).
  stcl> rmvault sync://localhost:2647/Projects/Sportster/top/*
```

rmversion

rmversion Command

NAME

```
rmversion           - Deletes versions from the vault
```

DESCRIPTION


```
This command deletes the specified version from the vault. You delete
versions from a vault, a process known as pruning, to free up disk
space. Use this command with caution; you cannot recover a deleted
version. This command does not affect files in your local work area.
```

```
You cannot delete:
  - Tagged versions (unless you use the -force option).
  - Version 1.1.
  - Version .1 when other versions exist on the branch
  - Version .1 when the version is upcoming. (For example, suppose you
    have a branch 1.4.1 that has no versions, but the branch is
    locked. In this case the upcoming version is 1.4.1.1, which
    cannot be deleted.)
  - Branch-point versions (for example, if 1.2.1 is a branch, you
    cannot delete version 1.2).
  - The Latest version on a locked branch (for example, if
    someone checks out version 1.3 with a lock, you
    cannot delete version 1.3 from the vault until the lock is
    released).

Use version-extended names to specify a version. A
version-extended name consists of a filename followed
by a semicolon and a version number or tag name (for example,
top.v;1.2 or top.v;rel13).
 Notes:
  - You cannot use wildcards (such as '*') when using
    version-extended names.
  - When in stcl/stclc mode, you must surround version-extended
    names (or any URL with a semicolon) with double quotes.
  - DesignSync does not reuse version numbers once they have
    been deleted from the vault. For example, assume the vault
    contains top.v;1.1, top.v;1.2, and top.v;1.3, and you
    use rmversion to delete top.v;1.2 and top.v;1.3. If you or
    another user later creates a new version of top.v
    (ci -new top.v), DesignSync names the new version top.v;1.4.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

This command supports the command defaults system.
```

Notes for Modules (Module-based)

```
You cannot remove the latest version of a module branch, even with the
-force option, unless the entire branch is deleted.

You cannot use wildcards (such as '*') when using module instance
names as your argument.

You cannot specify module instance names in a version extended
format.
```

Removing Orphaned Module Members (Module-based)

```
Module member versions are automatically removed when they are no
longer referenced by a module version. However, because module
members are stored as vaults, the rules governing vault version
```

removal also apply to member versions. Consequently, it is possible that a module member version no longer referenced by any module version cannot be removed until other module member versions are also removed (for example, it could be a branch-point version). Such member versions are known as orphans.

Later purge and rmversion operations may eliminate these barriers making it possible to remove these orphans. However, under normal rmversion operation, orphans will never be identified as candidates for removal because only member versions referenced by module versions being deleted are identified as candidates for removal.

Using the -scrub option to the rmversion command, you can remove all orphaned module member versions from the module. The -scrub option to rmversion searches through the entire module history and removes any orphaned module member versions.

SYNOPSIS

```
rmversion [-[no]force] [-report <mode>] [-scrub] [-trigarg <arg>]
          [--] <version> [<version> [...]]
```

ARGUMENTS

- DesignSync Object
- Server Module URL (Module-based)
- Workspace Module (Module-based)

Specify one or more of the following arguments:

DesignSync Object

```
<DesignSync object> Removes the specified DesignSync object from the
                    server. This object can be a version-extended
                    vault URL, or an object in your workspace.
```

Server Module URL (Module-based)

```
<module URL>        URL of the module containing the versions being
                    removed. Specify the URL as follows:
                    sync[s]://<host>[:<port>]/Modules/[<Category>...]
                     /<module>;<version>
                    where <host> is the SyncServer on which the
                    module resides, <port> is the SyncServer
                    port number, <Category> identifies the
                    optional category path, <module> is the name
```

of the module, and <version> is the version extension.

Note: You must specify the module URL as a version extended object.

Workspace Module (Module-based)

<workspace module>	Specifies the module instance name or path of the module containing the version being removed. By default, this will remove from the branch currently populated in the workspace. Note: The rmversion command accepts version-extended workspace folder and file paths. It does not accept version-extended module instance names. Also module instance names cannot be specified by using wildcard characters.

OPTIONS

-force (Module-based)

-[no]force	Determines whether you can delete tagged versions from the vault. -noforce does not delete tagged versions.(Default) -force deletes tagged versions. Use this option with caution because deleting a tagged version changes (possibly damaging) a configuration. Note: The Latest version of a module branch will never be removed unless the entire module branch is removed. Also the initial (1.1) version of a module cannot be removed.

-force (File-based)

-[no]force	Determines whether you can delete tagged versions from

```
the vault.

-noforce does not delete tagged versions.(Default)

-force deletes tagged versions. Use this option with
caution because deleting a tagged version changes
(possibly damaging) a configuration.
```

-report (Module-based)

```
-report <mode>
            Specifies the amount of output generated by the
            rmversion operation.

            Available modes are:
            o brief - Displays error messages. (Note: This mode
              does not display versions successfully deleted.)
            o normal - (the default mode) Displays:
              - The name and version number of each version
                deleted.
              - Error messages.
            o verbose - Displays:
              - For vault objects, the full vault URL path of each
                version being deleted.
              - The name and version number of each version
                deleted.
              - Error messages.
            Note: For module objects, data about individual module
            member versions is not displayed.  The command summary
            at the end of the command output indicates how many
            module member versions were removed.
```

-report (File-based)

```
-report <mode>
            Specifies the amount of output generated by the
            rmversion operation.

            Available modes are:
            o brief - Displays error messages. (Note: This mode
              does not display versions successfully deleted.)
            o normal - (the default mode) Displays:
              - The name and version number of each version
                deleted.
              - Error messages.
            o verbose - Displays:
              - For vault objects, the full vault URL path of each
                version being deleted.
              - The name and version number of each version
                deleted.
              - Error messages.
```

-[no]scrub (Module-based)

```
-[no]scrub    Specifies whether to remove any module member versions
              no longer referenced for any module versions; orphaned
              module members.
              -noscrub does not search for or remove any orphaned
              module members. (Default)
              -scrub expands the scope of the rmversion command to
              search for any orphaned module versions.
```

-trigarg

```
-trigarg <arg> Specifies an argument to be passed from the
               command line to the triggers set on the
               delete version operation. If the argument contains
               whitespace, enclose the argument within
               double quotation marks ("") if using the dss
               command shell or braces ({}) if using the stcl
               command shell.
```

--

```
--             Indicates that the command should stop looking for
               command options. Use this option when arguments
               to the command begin with a hyphen (-).
```

RETURN VALUE

```
none
```

SEE ALSO

```
remove, rmfile, rmfolder, rmvault, retire, tag, command defaults
```

EXAMPLES

- Example of Removing a File Version
- Example of Removing a File Specified with a Path
- Example of Removing Multiple Files with Associated Tags

Example of Removing a File Version

```
Delete version 1.2 of top.v, where top.v is in my current work area.
The double quotes are required in stcl/stclc mode.
```

```
  stcl> rmversion "top.v;1.2"
```

Example of Removing a File Specified with a Path

```
Delete version top.v;1.2 specifying an absolute path to the
file in the local work area. In dss/dssc mode, the quotes are optional.
  dss> rmversion "/home/Projects/ASIC/top.v;1.2"
```

Example of Removing Multiple Files with Associated Tags

```
Delete two versions of top.v, both of which have tags associated
with them.
  dss> rmversion -force top.v;1.2 top.v;rel13
```

select

select Command

NAME

```
select              - Identifies specific objects to be processed
```

DESCRIPTION

```
This command builds a list of objects on which commands can
operate. You might use a select list when you are going
to perform multiple operations on the same set of objects.
Many commands that accept objects as command arguments
support the '-selected' option. When you specify '-selected',
the command operates on this pre-built select list in addition to
any objects you specify as arguments.

You can specify wildcards when selecting objects. Use the
'unselect' command to remove objects from the select list.

Commands that operate on a select list can also operate on
objects you select from the DesignSync graphical interface.
Select one or more objects from the List View, then enter a
command from the command bar.
Notes:
  o The "Synchronize graphical and command-line interfaces "
    option from Tools->Options->GUI Options must be selected.
  o Selecting objects graphically clears your current select list.
```

SYNOPSIS

```
select [--] {-show | <argument> [<argument>...]}
```

ARGUMENTS

- Server Module (Module-based)
- Workspace Module (Module-based)
- Module Member (Module-based)
- DesignSync Object (File-based)

Server Module (Module-based)

```
<server module>     Server modules can be selected using URL of the
                    module.
                    sync[s]://<host>[:<port>]/<vaultPath> where
                    <host> is the SyncServer on which the module
                    resides, <port> is the SyncServer port number,
                    and <vaultPath> identifies the module to select.
```

Workspace Module (Module-based)

```
<workspace module>  Workspace modules can be selected.
```

Module Member (Module-based)

```
<workspace module   Workspace module members can be selected.
 member>

                  Note: Server module members, member versions,
                  branches, and hrefs do not have a specific server
                  address and therefore cannot be specified in a
                  selector list.
```

DesignSync Object (File-based)

```
<DesignSync object> Most DesignSync objects can be selected.
<DesignSync folder>
```

OPTIONS

- -show
- --

-show

```
 -show    Lists the objects in your select list.
```

--

```
 --       Indicates that the command should stop looking for
          command options. Use this option when arguments
          to the command begin with a hyphen (-).
```

RETURN VALUE

```
In dss/dssc mode, you cannot operate on return values, so the return
value is irrelevant.

In stcl/stclc mode, there is no return value except when you specify
-show, in which case the return value is a Tcl list of the
objects in your select list.
```

SEE ALSO

```
unselect, cancel, ci, co, ls, populate, tag, vhistory
```

EXAMPLES

- Example of Using Select on the Command Line to Select Files
- Example of Using Select within a Script

Example of Using Select on the Command Line to Select Files

```
This example selects all files that begin with 'samp' or have a
'.mem' extension, then checks out the selected files and 'top.v'.
Note that 'samp.mem' matches both the arguments to the select
command but is stored only once in the select list.
  dss> select samp* *.mem
  Already Selected: c:\Projects\Sportster\code\samp.mem
  dss> select -show
  file:///c|/Projects/Sportster/code/samp.asm
  file:///c|/Projects/Sportster/code/samp.lst
  file:///c|/Projects/Sportster/code/samp.mem
  file:///c|/Projects/Sportster/code/samp.s19
  file:///c|/Projects/Sportster/code/sample1.asm
  file:///c|/Projects/Sportster/code/test.mem
  dss> co -selected -lock -nocomment top.v
```

Example of Using Select within a Script

```
This example runs an stcl script called select.tcl, which
displays a message for each object in a directory with a '.v'
extension.
   # -- script start --
   select *.v
   foreach obj [select -show] {
       puts "$obj is selected."
   }
   # -- script end --

   stcl> run ./select.tcl
   file:///c|/Projects/Sportster/top/alu/alu.v is selected.
   file:///c|/Projects/Sportster/top/alu/mult8.v is selected.

   stcl>
```

setowner

setowner Command

NAME

```
setowner            - Sets the owner on the object specified
```

DESCRIPTION

```
This command sets the ownership of an object to the name specified.
The object can be project, project configuration, DesignSync vault
branch or module branch.

The owner of a branch is the creator of the initial version of the
branch unless a different owner is specified with the
setowner command. For example, the default owner of the main branch
(branch 1) is the creator of version 1.1 . The owner of an object's
main branch is also, by definition, the owner of the object's vault.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.
```

SYNOPSIS

```
setowner [--] <argument> <owner>
```

OPTIONS

- --

--

```
--          Indicates that the command should stop looking for
            command options. Use this option when arguments
            to the command begin with a hyphen (-).
```

RETURN VALUE

```
none
```

SEE ALSO

```
url owner, switchlocker
```

EXAMPLES

- Example of Setting the Ownership for a Project
- Example of Setting the Owner of a Branch

Example of Setting the Ownership for a Project

```
This example sets the ownership for the project
ASIC to 'johndoe':
  dss> setowner sync://myserver:myport/Projects/ASIC johndoe
```

Example of Setting the Owner of a Branch

```
This example sets the owner of the main branch of reg5.v to barbg:
  dss> setowner "sync://holzt:2647/Projects/Sportster/decoder/reg5.v;1"
barbg
```

switchlocker

switchlocker Command

NAME

```
switchlocker        - Changes the current owner of a lock
```

DESCRIPTION

```
This command changes the lock owner of a branch. This command is
particularly useful when two or more people are working on the same
branch.

The following design scenario highlights the function of switchlocker.

  UserA and UserB share the same work area and will be editing
  the same design object. UserA checks out the object for
  editing, thereby locking the branch. The object is modified by
  UserA or UserB, or both (assuming the proper permissions have
  been set on the object). UserB then needs to check in the
  changes (maybe UserA is unavailable to perform the
  checkin). UserB can use the switchlocker command to take lock
  ownership and then perform the checkin.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

This command supports the command defaults system.
```

SYNOPSIS

```
switchlocker [-modulecontext <context>] [--] <locker> <argument>
```

ARGUMENTS

- Username of New Locker
- Server Module Branch (Module-based)
- Module Member Argument (Module-based)
- DesignSync Object (File-based)

Username of New Locker

```
<locker>              Username of the new locker of the file.
```

Server Module Branch (Module-based)

```
<server module branch> Specifies the locked module branch being
                       switched.

                       Note: Any locked objects in the branch must be
                       held by the same user who holds the branch
                       lock.  If there are locked objects in the
                       branch held by a different user, you must use
                       the unlock command to unlock those objects.
```

Module Member Argument (Module-based)

<module member>	Specifies the locked module member to switch. When you specify an individual module member, you must use the -modulecontext option to specify the module context appropriate for the module member.

DesignSync Object (File-based)

<DesignSync object>	Specifies the object being switched. If the object is on a branch or is not the current version, you must specify the branch or version information for the object.

OPTIONS

- -modulecontext (Module-based)
- --

-modulecontext (Module-based)

-modulecontext <context>	Identifies the module version of the objects being switched to a different locker. Specify the module context with the sync URL of the desired module. For example: sync://server1:2647/Modules/Chip;RelA Note that you cannot use a -modulecontext option to operate on objects from more than one module; the -modulecontext option takes only one argument, and you can use the -modulecontext option only once on a command line. When the modulecontext option is used, the argument must specify the natural path of the object being switched.

--

--	Indicates that the command should stop looking for command options. Use this option when the object you specify begins with a hyphen (-).

RETURN VALUE

```
none
```

SEE ALSO

```
cancel, unlock, setowner, url properties, command defaults
```

EXAMPLES

- Example of Switching the Locker for a Module Member (Module-based)
- Example of Switching the Locker for a DesignSync File-Basd Objects (File-based)

```
The following examples shows how two users with a shared
work area might use switchlocker. User 'goss' must take over the
lock from 'barbg' before 'goss' can check in the file.
```

Example of Switching the Locker for a Module Member (Module-based)

```
This example shows using switchlocker on a module object, chip.c.
Note that the module object being specified is proceeded by a leading
slash (/).  This means that the objects is in the module base
directory (Chip/.)

dss> showlocks Chip%0
 Module Chip, branch 1 (Trunk) has content locks:

 User    Date               Name     Where
 ----    ----               ----     -----
 barbg 11/13/2006 15:59   /chip.c  /home/barbg/chip/chip.c
 ...

dss> switchlocker -modulecontext \
 sync://srvr2.ABCo.com:2647/Modules/Chip;1.7 goss /chip.c

 sync://srvr2.ABCo.com:2647/Modules/Chip;1
     /chip.c : Success

dss> showlocks sync://srvr2.ABCo.com:2647/Modules/Chip

 Module Chip, branch 1 (Trunk) has content locks:

 User    Date              Name    Where
 ----    ----              ----    -----
 goss  11/13/2006 15:59  /chip.c   Unknown
 ...

Note:  The Where value is unknown because the lock is no longer
associated with the original workspace.
```

Example of Switching the Locker for a DesignSync File-Basd Objects (File-based)

```
This example shows using switchlocker on a DesignSync file-based
object, top.v.

  dss> ls -report RU top.v
  Version          Locked By   Name
  -------          ---------   ----
  1.2 -> 1.3       barbg*      top.v

  dss> ci -nocomment top.v

  Beginning Check in operation...

  Checking in: top.v   : Failed:som: Error 102: Locked By Other User.

  dss> switchlocker goss top.v
  SwitchLocker:       success
  dss> ls -report RU top.v
  Version          Locked By   Name
  -------          ---------   ----
  1.2 -> 1.3       goss*       top.v
  dss> ci -nocomment top.v

  Beginning Check in operation...

  Checking in: top.v    : Success - New version: 1.3
  dss>
```

unlock

unlock Command

NAME

```
unlock              - Releases the lock on the specified object(s)
```

DESCRIPTION

- Notes on Modules (Module-based)
- Note on File-Based Objects (File-based)
- Auto-Branching (File-based)

```
This command releases the lock on a specified object(s). This command
is used primarily to release object locks on the server. To release a
lock on an object you have checked out in your work area, use the
'cancel' command instead of 'unlock'. Use the 'unlock' command to
remove a lock held by someone else, or if you no longer have the
object that you checked out in your work area.
```

Only one user can have a lock on a given branch of an object at a time. Having a lock prohibits other users from checking in changes to that branch; however, other users (or the same user in different work areas) can independently lock, unlock, and check in changes to other branches.

To remove a lock and change states, use the 'cancel' command. Also, if you have a lock taken away from you by another user (with the 'unlock' command), you should cancel your checkout (with the 'cancel' command) to return your local object to a consistent state.

Unlock is equivalent to performing 'cancel -keep' on an object because unlock does not affect the local copy of the file in the work area. The unlock action replaces locked references in the workspace with copies.

This command is subject to access controls on the server. See the ENOVIA Synchronicity Access Control Guide for details.

This command supports the command defaults system.

Notes on Modules (Module-based)

You lock a module branch by using the -lock command.

Filter and exclude lists are used to include or exclude objects to be unlocked. Filter lists are used to include or exclude module objects or to include DesignSync objects. Exclude lists are used to exclude DesignSync objects.

Note: Regardless of whether -filter or -exclude is used to exclude an object, the command output message indicates that the object was "excluded by filter."

The natural path argument for a module, as shown in the example, always begins with a "/" character.

Note: For module members, the locker keyword is always a null value, so unlock does not update the keyword in module members when the lock is released.

Note on File-Based Objects (File-based)

You lock a branch by checking out an object with the -lock option to the co, populate, or ci command, or by using the lock command on a module branch.

Exclude lists are used to exclude objects from the unlock. Filter lists are used to include or exclude module objects or to include DesignSync objects. Exclude lists are used to exclude DesignSync objects.

```
Note: When -exclude is used to exclude an object, the command output
message indicates that the object was "excluded by filter."
```

Auto-Branching (File-based)

```
You can create a new, locked branch by using 'co -lock' with a selector
and autobranching. This branch can be unlocked without creating a new
version by:

 - Using 'cancel' from the workspace where the branch was locked.
 - Using 'unlock' on the vault.
 - Using ci from the workspace where the branch was locked, without
   making modifications.

In these cases, the lock is removed from the vault, the auto-created
branch is removed, and the branch tag is deleted. If the branch is
removed but still exists in the metadata of a workspace, some commands
(such as the 'url' commands and 'vhistory') will fail with "No such
version.
```

SYNOPSIS

```
unlock [-branch <branch> | -branch auto(<branch>)]
       [-exclude <object>[,<object>...]]
       [-modulecontext <context>] [-[no]recursive] [-[no]selected]
       [-trigarg <arg>] [--] [<argument> [<argument> ...]]
```

ARGUMENTS

- Module Branch/Module Version (Module-based)
- Module Member (Module-based)
- Module Folder (Module-based)
- DesignSync Object (File-based)
- DesignSync Folder (File-based)
- DesignSync Vault (File-based)

Module Branch/Module Version (Module-based)

```
<module branch|        Specify a server module branch or version
 module version>       to remove from the lock from the server
                       version, or or a workspace module to remove
                       the lock from the associated branch of that
                       module, in the workspace and on the server.

                       The natural path to a server module must
                       begins with "/"
```

Module Member (Module-based)

<module member>	Specify a module member to remove the lock in the server and the workspace. The server can be specified with the -modulecontext option. If the -modulecontext option is not used, the command derives the module context from the persistent selector of the workspace.

Module Folder (Module-based)

<module folder>	Specify a module folder to remove the locks from all objects in the folder. If the -modulecontext option is not used, the command derives the module context from the persistent selector of the workspace. If you unlock a server module folder, you must specify the natural path beginning with the "/" character.

DesignSync Object (File-based)

<DesignSync object>	A versionable file or collection object, in which case the current branch is unlocked. Note: If you specify a collection member as the object to be operated on, DesignSync skips the object and warns that the object is not versionable. If DesignSync attempts to operate on a collection member specified implicitly (through the use of wildcards or a recursive operation), DesignSync silently skips the object. You can change this behavior by using the SyncAdmin "Map operations on collection members to owner" setting. If you select this setting and DesignSync attempts to operate on a collection member during a revision control operation, DesignSync determines the member's owner collection and operates on the collection as a whole.

DesignSync Folder (File-based)

<DesignSync folder>	Specify a DesignSync folder on the server or in your workspace (local) to unlock all objects in the folder. To unlock all objects in sub-folders of the specified folder, use the -recursive option.

DesignSync Vault (File-based)

```
<DesignSync vault>      Specify a DesignSync vault to unlock the
                        initial branch of the objects in the vault.
```

OPTIONS

-branch (Module-based)

```
-branch <branch>        Unlocks the branch specified by the branch or
                        version tag, or branch numeric.  By default
                        (without -branch), the current branch of each
                      specified object is unlocked. This option
                      overrides the object's persistent selector
                      list. If <branch> resolves to a version, the
                      branch of that version is unlocked.

                        Note: The -branch option accepts a single
                        branch tag, a single version tag, or a branch
                        numeric. It does not accept a selector or
                        selector list.

                        Note: The -branch option is ignored when the
                        module branch information is specified by the
                        server URL argument.
```

-branch (File-based)

```
-branch <branch>        Unlocks the branch specified by the branch or
| -branch               version tag, auto-branch selector, or branch
  auto(<branch>)        numeric.  By default (without -branch), the
                        current branch of each specified object is
                        unlocked. This option overrides the object's
                        persistent selector list. If <branch> resolves
                        to a version, the branch of that version is
                        unlocked.
```

Note: The -branch option accepts a single branch tag, a single version tag, a single auto-branch selector tag, or a branch numeric. It does not accept a selector or selector list.

-exclude

-exclude <objects>

Specifies a comma-separated list of objects to exclude from the operation. Wildcards are allowed.

Do not specify paths in your arguments to -exclude. Before operating on each object (such as during a recursive unlock operation), DesignSync compares the object's leaf name (path stripped off) to the exclude list to see if there is a match. Because the object's path is removed, the object will not match any object in the exclude list specified with a path. For example, if you specify '-exclude bin/*.exe', you will not successfully exclude bin/foo.exe or any other *.exe file. You need to instead specify '-exclude *.exe', or '-exclude foo.exe' if you want to exclude only 'foo.exe'. This means, however, that you cannot exclude a specific instance of an object -- you exclude all matching objects.

In addition to objects you specify using the -exclude option, the "These objects are always excluded" field from the Tools->Options->General->Exclude Lists dialog box in the DesignSync graphical user interface lists objects that are always excluded from revision-control operations.

-modulecontext (Module-based)

-modulecontext <context>

Specifies the server module branch in which the objects are being unlocked.

You can specify a server module URL and the branch or version Id, (sync://server1:2647/Modules/Chip;RelA) or specify the module as a module instance.

Note: You cannot use a -modulecontext option to operate on objects from more than one module; the -modulecontext option takes only one argument, and you can use the

-modulecontext option only once on a command line.

-[no]recursive (Module-based)

-[no]recursive	Determines whether to unlock the objects in the specified folder or all objects in the folder and all objects in the subfolders. This option is ignored if the argument is not a module. -norecursive removes locks only from objects in the specified folder. (Default) -recursive removes the locks from the objects in the specified folder and all subfolders. Note: On GUI clients, -recursive is the initial default.

-[no]recursive (File-based)

-[no]recursive	Determines whether to unlock the objects in the specified folder or all objects in the folder and all objects in the subfolders. This option is ignored if the argument is not a DesignSync folder. -norecursive removes locks only from objects in the specified folder. (Default) -recursive removes the locks from the objects in the specified folder and all subfolders. Note: On GUI clients, -recursive is the initial default.

-[no]selected

-[no]selected	Determines whether the operation is performed just on the objects specified at the command line or on objects specified at the command line and objects in the select list (see the 'select' command) -noselected unlocks only objects specified on the command line. (Default) -selected unlocks objects specified on the command and in the select list. Note: If no objects are specified on the command line, the -selected option is implied.

-trigarg

```
-trigarg <arg>          Specifies an argument to be passed from the
                        command line to the triggers set on the
                        unlock operation. If the argument contains
                        whitespace, enclose the argument within
                        double quotation marks ("") if using the dss
                        command shell or braces ({}) if using the stcl
                        command shell.
```

--

```
--                      Indicates that the command should stop looking
                        for command options. Use this option when
                        arguments to the command begin with a hyphen
                        (-).
```

RETURN VALUE

```
The command has no Tcl return value.

The command does provide a list of the objects processed by the
command and information about whether the command succeeded, failed,
or was skipped.

Note: If an object was "excluded by filter," it may have been
excluded either with the -filter option (for modules) or with the
-exclude option (for DesignSync objects.)
```

SEE ALSO

```
cancel, ci, co, lock, populate, select, selectors, switchlocker,
command defaults
```

EXAMPLES

- Example of Unlocking a Module Member in the Workspace (Module-based)
- Example of Unlocking a Module Member Using -modulecontext (Module-based)
- Example of Unlocking Specific Files (File-based)
- Example of Unlocking the Contents of a Directory Recursively (File-based)

Example of Unlocking a Module Member in the Workspace (Module-based)

```
This example unlocks the "Chip.doc" module member on the Trunk branch
of the Chip module, as defined by the persistent selector in the
workspace. The "Chip.doc" file is located in a doc subdirectory
```

```
within the module.

dss> unlock /Doc/Chip.doc
 Beginning Unlock operation...

 Unlocking objects in module Chip%0 with base dir
 c:\workspaces\chip\ ...

 /Doc/Chip.doc: Unlocked
```

Example of Unlocking a Module Member Using -modulecontext (Module-based)

```
This example unlocks the "Chip.doc" module member on the Trunk branch
of the Chip module. The "Chip.doc" file is located in a doc
subdirectory within the module.

Note: When you specify a module or module member to unlock, you must
specify the natural path to the specified argument.

dss> unlock -modulecontext sync://host:2647/Modules/Chip;Trunk \
     /Doc/Chip.doc
 Beginning Unlock operation...

 Unlocking:   sync://serv1.ABCo.com:2647/Modules/Chip;1 :
     /doc/Chip.doc: Unlocked

 Unlock operation finished.

  {Objects succeeded (1)} {}
```

Example of Unlocking Specific Files (File-based)

```
This example unlocks the 'Rel2.1' branch of the 'alu.v' and
'decoder.v' files.
  dss> unlock -branch Rel2.1 alu.v decoder.v
```

Example of Unlocking the Contents of a Directory Recursively (File-based)

```
In the following example, a developer went on vacation while having
many of the files in the 'code' vault folder locked. The following
command recurses the 'code' vault folder and removes the
locks.
  dss> unlock -rec sync://host:2647/Projects/Sportster/code

  Beginning Unlock operation...

  Unlocking: sync://host:2647/Projects/Sportster/code/samp.asm; : Not locked
  Unlocking: sync://host:2647/Projects/Sportster/code/samp.lst; : Unlocked.
  Unlocking: sync://host:2647/Projects/Sportster/code/test.mem; : Unlocked.
  Unlocking: sync://host:2647/Projects/Sportster/code/test.asm; : Not locked
```

```
Unlocking: sync://host:2647/Projects/Sportster/code/samp.mem; : Unlocked.
Unlocking: sync://host:2647/Projects/Sportster/code/test.lst; : Not locked
Unlocking: sync://host:2647/Projects/Sportster/code/test.s19; : Unlocked.

Unlock operation finished.
```

unselect

unselect Command

NAME

```
unselect            - Removes files from the 'selected' list
```

DESCRIPTION

```
This command removes specified files from the list of selected
objects. Many commands that accept filenames also support the
-selected option, which feeds this pre-built list of files to the
command for processing. As with most commands that accept
filenames, wildcard file specifications are also supported.
```

SYNOPSIS

```
unselect [-quiet] [-all | [--] <argument> [<argument>...]]
```

ARGUMENTS

- Server Module (Module-based)
- Workspace Module (Module-based)
- Workspace Module Member (Module-based)
- DesignSync Object (File-based)

Server Module (Module-based)

```
<server module>     Server modules can be selected using the URL of
                    the module.
                    sync[s]://<host>[:<port>]/<vaultPath> where
                    <host> is the SyncServer on which the module
                    resides, <port> is the SyncServer port number,
                    and <vaultPath> identifies the module to select.
```

Workspace Module (Module-based)

```
<workspace module>  Workspace modules can be selected.
```

Workspace Module Member (Module-based)

```
<workspace module   Workspace module members can be selected.
 member>

                    Note: Server module members, member versions,
                    branches, and hrefs do not have a specific server
                    address and therefore cannot be specified in a
                    selector list.
```

DesignSync Object (File-based)

```
<DesignSync object> Most DesignSync objects can be selected.
<DesignSync folder>
```

OPTIONS

- -all
- -quiet
- --

-all

```
-all     Remove all objects from the select list.

         This option is mutually exclusive with specifying an
         argument to this command.
```

-quiet

```
-quiet   Do not report the names of objects being deselected.
```

--

```
--       Indicates that the command should stop looking for
         command options. Use this option when arguments
         to the command begin with a hyphen (-).
```

RETURN VALUE

```
none
```

SEE ALSO

```
select, cancel, ci, co, ls, tag, populate
```

EXAMPLES

- Example of Removing Specified Objects from the Select List
- Example of Removing All Objects from the Select List

Example of Removing Specified Objects from the Select List

```
This example removes foo1.v and all files that match bar*.v from
the select list:
  dss> unselect foo1.v bar*.v
```

Example of Removing All Objects from the Select List

```
This example removes all objects from the select list:
  dss> unselect -all
```

upload

upload Command

NAME

```
upload              - Upload/Update compressed IP stored in DesignSync
```

DESCRIPTION

- Understanding How a Temporary Directory is used for Upload
- Order of Precedence for Temp Directory:

```
The command allows you to upload or update a tar or gzipped tar
archive to DesignSync in an efficient manner so that, instead of
replacing the archive with the next version, DesignSync updates
only the elements within the archive file that have changed from the
previous version.

By performing a change (delta) calculation and only checking in the
```

```
 changed object set, DesignSync provides both improved speed during
 checkin and checkout and reduces the amount of disk space required
 for storing the IP.

 The user running the upload should examine the tar file to make sure
 it contains none of
 the following:
  o unnecessary or undesired parent directories
  o absolute path directories

 These should be removed before performing the upload.

Notes:
 o The executables (binaries) for tar or gtar must be on the user's
   path in order for the command to work.

 o DesignSync also provides a graphical user interface for uploading
   IP through the DesignSync Web Interface.  For more information, see
   the DesignSync Administrator's Guide.

 This command is subject to Access Controls on the server.

 This command supports the command defaults system.
```

Understanding How a Temporary Directory is used for Upload

```
 The compressed archive is exploded in a temporary directory and
 compared against the last version, if applicable, on the server and
 only the changed object set is checked in.

  Tip: For optimal operation, DesignSync recommends that the upload
       directory contain at least  2.5* the size of the uncompressed
       archive file.

 By default, this operation is performed in the temporary directory
 specified by the Upload_Tmp_Dir registry setting or the SYNC_TMP_DIR
 environment variable.  If neither of these is set, DesignSync uses the /tmp
 directory on the repository server. For more information on setting
 the Upload_Tmp_Dir registry setting, or the SYNC_TMP_DIR environment
 variable, see the DesignSync Administrator's Guide.

 You can optionally specify either a local directory or an alternate
 location on the server.  This is especially useful for servers where
 you cannot control the server space consumption; specifying an
 alternative disk partition or performing the delta comparison locally
 allows you to make sure you have enough space to perform the
 operation. Specifying an option on the command line overrides any
 existing settings.
```

Order of Precedence for Temp Directory:

```
 Note: DesignSync will use this order to determine which tmp
```

directory to use for the upload operation. If there is no set value, DesignSync will check the next location on this. If there is a value set, but DesignSync is unable to use it, for example, because of incorrect write permissions, the command will fail.

1. If the -vault option is used, and -servertmpdir or -localtmpdir is specified, the value of <tmpdir> is used. If the -workspace option is specified, the workspace is used as the tmp directory.

2. If the command defaults system is used to set a value -servertmpdir or -localtmpdir, that value is used as the tmp directory.

3. If the UploadTmpDir registry setting is specified, that value is used as the tmp directory.

4. If the SYNC_TMP_DIR environment variable is set on the server machine, that value is used as the tmp directory.

5. If the TMPDIR environment variable is set on the server machine, that value is used as the tmp directory.

6. If no other values are set, DesignSync uses the /tmp directory on the server machine.

SYNOPSIS

```
upload [-branch <branchname>] [-[no]collections]
       [-[no]comment <comment>] [-[no]new]
       [-report brief | normal | verbose] [-tag <tagname>]
       [-vault <vaulturl> [-servertmpdir <tmpdir>] |
       [-vault <vaulturl> [-localtmpdir <tmpdir>] |
       [ -workspace <path>] <tarfile>
```

ARGUMENTS

- Tar file

Tar file

<tarfile> Specify a tar or gzipped tar archive to upload or update on the server. The archive can be specified with an absolute or relative path. The file extension for the tar file must be either .tar or .tgz in order for DesignSync to recognize the file.

NOTE: If the tar file contains .SYNC directories, they are automatically ignored and not checked in with the archive.

OPTIONS

-branch

-branch <branchname>	Specifies the branch on which to place the archive. You can specify only one branch with this option. If no branch is specified, DesignSync uploads to the Trunk branch. You cannot specify a branch tag for the initial archive upload, which is always checked into the Trunk branch. For the <branchname>, specify a branch tag (for example, rel40) or branch numeric (for example, 1.4.2) only. Do not specify a colon (:) with the branchname. If a temp directory (other than the /tmp default) is specified for the upload, and the -branch option is used, the specified branch must already exist on the server. The -branch option is mutually exclusive with the -new option.

-[no]collection

-[no]collection	Specifies whether the compressed package includes collections objects. For more information on collection handling, see the DesignSync Administrator's Guide. -nocollection specifies that the compressed archive does not contain collection objects. This allows the upload process to use reference mode, improving the speed of operations. (Default) -collection specifies that the compressed archive

	contains collection objects. The upload process will not attempt to use reference mode which would process collections incorrectly.

-[no]comment

-[no]comment ["<comment>"]	Specifies whether a text description of the upload is stored with the checked in version. -nocomment performs the upload with no comment.(Default) -comment <text> stores the value of <text> as the module comment. To specify a multi-word comment, use quotation marks ("") around the comment text.

-localtmpdir

-localtmpdir <tmpdir>	When -vault is used, the -localtmpdir option is used to specify a tmp directory path on the local (client) machine to be used for the upload operation. When the upload is completed, any objects placed in this directory during upload are removed.

-[no]new

-[no]new	Performs the initial checkin of the archive. The initial archive checkin must be performed on the Trunk branch. -nonew is used to update the archive in revision control. If the archive does not exist and -nonew is selected, the command fails. (Default) -new is used to create or update the archive. If the archive exists and the -new option is specified, the archive is updated. The -new option is mutually exclusive with the -branch option.

-report (Module-based)

-report brief | normal| verbose	Controls the amount and type of information displayed by the command. brief mode reports the newly created module

version, along with the generated tag.

Normal mode additionally reports a list of the changes in the archive, including: added files, removed files and changed files.

Verbose mode is equivalent to normal mode.

-report (File-based)

-report brief | normal| verbose	Controls the amount and type of information displayed by the command. brief mode reports the generated tag. Normal mode additionally reports a list of the changes in the archive, including: added files, retired files and changed files. Verbose mode is equivalent to normal mode.

-servertmpdir

-servertmpdir <tmpdir>	When -vault is used, the -servertmpdir option is used to specify a tmp directory path on the repository server to be used for the upload operation. When the upload is completed, any objects placed in this directory during upload are removed.

-tag

-tag <tag>	Applies the specified tag to the data being imported. This tag can be used to get the data later, or example, when populating the archive into a workspace. If the tag already exists it moves to the new version. Note: An automatically generated tag, in the form Archive.<#> is also applied to the data being imported, where the initial value of # is 1, and then the number is incremented as archive is updated.

-vault (Module-based)

```
 -vault <vaultURL>          Specify the module URL and optionally a server
 [-servertmpdir <tmpdir>]   or local path to use as a  temporary upload
 | [-localtmpdir <tmpdir>] directory.

                        Specify the module URL in the format:

sync[s]://<host>:<port>/Modules/[<category>...]/<Module>

                        If the module does not exist, then it will be
                        created, if the command is run with the -new
                        option.

                        When you specify an alternate tmp directory for
                        upload, you can specify a server path on on the
                        repository server or a local path on the
                        client system. For more information on
                        specifying a server path, see the -servertmpdir
                        option. For more information on specifying a
                        local path, see the -localtmpdir option.

                        This option is mutually exclusive with
                        -workspace. Either -workspace or -vault must be
                        specified.
```

-vault (File-based)

```
 -vault <vaultURL>    Specify the vault URL and optionally a server or
   [-servertmpdir <tmpdir>] local path to use as a temporary upload
   | [-localtmpdir <tmpdir>] directory.

                        Specify the vault URL in the format:
                       sync[s]://<host>:<port>/[<Project>...]/<vault>

                        If the vault does not exist, then it will be
                        created, if the command is run with the -new
                        option.

                        When you specify an alternate tmp directory for
                        upload, you can specify a server path on on the
                        repository server or a local path on the
                        client system. For more information on
                        specifying a server path, see the -servertmpdir
                        option. For more information on specifying a
                        local path, see the -localtmpdir option.

                        This option is mutually exclusive with
                        -workspace. Either -workspace or -vault must be
                        specified.
```

-workspace

```
 -workspace             Specify an existing, unmodified workspace
```

```
<path>              as a staging area to unpack the new archive,
                    determine the changes necessary and send only
                    the changes to the server.  If this is used for
                    an initial upload, the archive is unpacked in
                    the workspace and the entire contents of the
                    archive is uploaded. For the initial upload,
                    DesignSync uses the persistent selector to
                    determine the module/vault for checkin.

                    This is a performance enhancement that minimizes
                    the server processing time needed to compute the
                    deltas by pre-computing the deltas in the
                    workspace.

                    The workspace must be owned and writable by the
                    person running the command.

                    The -workspace option is mutually exclusive
                    with -vault and -branch. The -workspace option
                    is only supported for UNIX workspaces.
```

RETURN VALUE

```
This command does not return any TCL values. DesignSync provides
status messages while the command runs.  If the command fails,
DesignSync returns an error explaining the failure.
```

SEE ALSO

```
defaults, access, ci
```

EXAMPLES

- Example of Performing an Initial Upload (Module-based)
- Example of Specifying a Server Temporary Directory for Module Upload (Module-based)
- Example of Specifying a Local Temporary Directory for Module Upload (Module-based)
- Example of Performing an Upload Using a Module Workspace (Module-based)
- Example of Performing an Initial Upload (File-based)
- Example of Performing an Upload Using a File-Based Workspace (File-based)
- Example of Specifying a Server Temporary Directory for File-based Upload (File-based)
- Example of Specifying a Local Temporary Directory for File-based Upload (File-based)

Example of Performing an Initial Upload (Module-based)

```
This example shows performing an initial upload to a module.
```

```
 Note: This example has been run in normal mode, which means that each
 object in the tar file is listed in the command output as it is
 checked in.  For brevity, those checkin lines have been removed.

 dss> upload -vault sync://qelwsun14:30126/Modules/IPWIP/FinalIP -new
  -comment "IP Finals version 1.0" FinalIP.tar

  Logging to /home/rsmith/dss_04012014_181455.log
  3DEXPERIENCE6R2022x

 Beginning Check in operation...

  Checking in objects in module FinalIP%0

  Total data to transfer: 7340 Kbytes (estimate), 626 file(s), 0
collection(s)
  Checking in:
  ...

  FinalIP%0: Version of module in workspace updated to 1.2

  Finished checkin of Module FinalIP%0, Created Version 1.2

  Time spent: 10.5 seconds, transferred 0 Kbytes, average data rate
  0.0 Kb/sec

  Checkin operation finished.

  NOTE: Workspace module argument 'FinalIP%0' supplied; will tag
  'sync://serv1.ABCo.com:2647/Modules/IPWIP/FinalIP;1.2'

  Beginning module tag operation on 'sync://qelwsun14:30126' ...

  Tagging:     sync://serv1.ABCo.com:2647/Modules/IPWIP/FinalIP;1.2 :
  Added tag 'Archive.1' to version '1.2'

  Module tag operation finished on 'sync://serv1.ABCo.com:2647'.
```

Example of Specifying a Server Temporary Directory for Module Upload (Module-based)

```
 This example updates an IP checked into a module.  It uses a
 specified directory on the server as its temporary storage area
 rather than the server default which allows you to make sure that the
 space you need  for the operation is available.

 Note: This example has been run in normal mode, which means that each
 object in the tar file is listed in the command output as it is
 populated, checked in, and tagged.  For brevity, the individual
 object detail lines have been removed.

   dss> upload -vault sync://serv1.ABCo.com:2647/Modules/CustomerIP
  -servertmpdir /home/syncadmin/tmp -comment "Uploaded IP"
  ./FinalIP.tar
   Beginning Check in operation...
```

```
  Checking in: ...
  ...
  Checkin operation finished.

  Beginning Tag operation...
  ...
  Tag operation finished.
  dss>
```

Example of Specifying a Local Temporary Directory for Module Upload (Module-based)

```
 This example updates an IP checked into a module.  It uses a
 specified directory on the client machine as its temporary storage
 area rather than the server default which allows you to make sure
 that the space you need for the operation is available and reduces
 processing time on the server.

 Note: This example has been run in normal mode, which means that each
 object in the tar file is listed in the command output as it is
 populated, checked in, and tagged.  For brevity, the individual
 object detail lines have been removed.

    dss> upload -vault sync://serv1.ABCo.com:2647/Modules/CustomerIP
  -localtmpdir ~/tmpfiles -comment "Uploading new version" FinalIP.tar

  Beginning Check in operation...
  Checking in: ...
  ...
  Checkin operation finished.

  Beginning Tag operation...
  ...
  Tag operation finished.
  dss>
```

Example of Performing an Upload Using a Module Workspace (Module-based)

```
 This example updates an IP checked into a module.  It uses the module
 workspace as its temporary storage area rather than the server which
 reduces the processing time needed on the server.

 Note: This example has been run in normal mode, which means that each
 object in the tar file is listed in the command output as it is
 populated and checked in.  For brevity, the individual object detail
 lines have been removed.


upload -comment "uploading IP Finals version 1.5" -workspace
       ~rsmith/MyMods/customerIP ../FinalIP.tar

  Beginning populate operation at Wed Apr 02 10:45:54 AM EDT 2014...
```

```
  Populating objects in Module          FinalIP%0
                        Base Directory  /home/rsmith/MyMods/customerIP
                        Without href recursion

  Fetching contents from selector 'Trunk:', module version '1.2'
  ... [Fetching List of Objects in Lock Mode]


  FinalIP%0 : Version of module in workspace retained as 1.2

  Finished populate of Module FinalIP%0 with base directory
/home/rsmith/MyMods/customerIP

  Time spent: 0.0 seconds, transferred 0 Kbytes, copied from local
  cache 0 Kbytes, average data rate 0.0 Kb/sec

  Finished populate operation.

  Beginning Check in operation...

 Checking in objects in module FinalIP%0

 Total data to transfer: 7102 Kbytes (estimate), 596 file(s), 0 collection(s)
  Progress: 0 Kbytes, 0 file(s), 0 collection(s), 0.0% complete
  Progress: 1 Kbytes, 0 file(s), 0 collection(s), 0.0% complete
  Progress: 4975 Kbytes, 404 file(s), 0 collection(s), 68.1% complete
  Progress: 7259 Kbytes, 596 file(s), 0 collection(s), 100.0% complete

  ... [Checking in new files, removing locks]

  FinalIP%0: Version of module in workspace updated to 1.3


  Finished checkin of Module FinalIP%0, Created Version 1.3

  Time spent: 15.7 seconds, transferred 7259 Kbytes, average data rate 463.8
Kb/sec

  Checkin operation finished.


  NOTE: Workspace module argument 'FinalIP%0' supplied; will tag
  'sync://serv1.ABCo.com/Modules/IPWIP/FinalIP;1.3'

  Beginning module tag operation on 'sync://serv1.ABCo.com:2647' ...

  Tagging:     sync://serv1.ABCo.com:2647/Modules/IPWIP/FinalIP;1.3 :
  Added tag 'Archive.2' to version '1.3'

  Module tag operation finished on 'sync://serv1.ABCo.com:2647'.
```

Example of Performing an Initial Upload (File-based)

This example shows performing an initial upload to a file-based vault.

```
Note: This example has been run in normal mode, which means that each
object processed in the tar file is listed in the command output.
For brevity, these lines have been removed.

dss> upload -comment "IP rel 1.0 handoff" -vault
sync://serv1.ABCo.com:2647/Projects/customerIP -new FinalIP.tar

 Operation continuing, please wait...
 sync://serv1.ABCo.com:2647/Projects/customerIP  Success Folder Made
 Logging to /home/rsmith/dss_04042014_123427.log
 3DEXPERIENCE6R2022x

 Beginning Tag operation...

 ... [List of tag files removed]

Tag operation finished.
```

Example of Performing an Upload Using a File-Based Workspace (File-based)

```
This example updates an IP checked into a file-based vault.  It uses a
workspace as its temporary storage area rather than the server which
reduces the processing time needed on the server.

Note: This example has been run in normal mode, which means that each
object in the tar file is listed in the command output as it is
populated, checked in, and tagged.  For brevity, the individual
object detail lines have been removed.

dss> upload -comment "IP rel 2.0 handoff" -workspace
~rsmith/workspaces/customerIP FinalIP.tar

 Beginning populate operation at Fri Apr 04 02:22:56 PM EDT 2014...
 ...

 Populated '/home/rsmith/workspaces/customerIP'

 Finished populate operation.

 Beginning Check in operation...
 ...

 Checkin operation finished.

 Beginning Tag operation...
 ...

 Tag operation finished.
```

Example of Specifying a Server Temporary Directory for File-based Upload (File-based)

```
This example updates an IP checked into a file-based vault.  It uses a
specified directory on the server as its temporary storage area
rather than the server default which allows you to make sure that the
space you need  for the operation is available.

Note: This example has been run in normal mode, which means that each
object in the tar file is listed in the command output as it is
populated, checked in, and tagged.  For brevity, the individual
object detail lines have been removed.

 dss> upload -vault sync://serv1.ABCo.com:2647/Projects/CustomerIP
-servertmpdir /home/syncadmin/tmp -comment "Uploaded IP"
./FinalIP.tar
  Beginning Check in operation...
 Checking in: ...
 ...
 Checkin operation finished.

 Beginning Tag operation...
 ...
 Tag operation finished.
 dss>
```

Example of Specifying a Local Temporary Directory for File-based Upload (File-based)

```
This example updates an IP checked into a file-based vault.  It uses a
specified directory on the client machine as its temporary storage
area rather than the server default which allows you to make sure
that the space you need for the operation is available and reduces
processing time on the server.

Note: This example has been run in normal mode, which means that each
object in the tar file is listed in the command output as it is
populated, checked in, and tagged.  For brevity, the individual
object detail lines have been removed.

  dss> upload -vault sync://serv1.ABCo.com:2647/Projects/CustomerIP
 -localtmpdir ~/tmpfiles -comment "Uploading new version" FinalIP.tar

 Beginning Check in operation...
 Checking in: ...
 ...
 Checkin operation finished.

 Beginning Tag operation...
 ...
 Tag operation finished.
 dss>
```

Navigational

cd

cd Command

NAME

```
cd                  - Changes your current directory
```

DESCRIPTION

```
This command is the standard Tcl 'cd' command. It lets you change
your current directory as viewed by the operating system. You can
specify a relative or absolute path. Specifying 'cd' without an
argument puts you in your home directory (as defined by $HOME on
UNIX or your user profile, which is managed by the User Manager
tool, on Windows).

In general, use 'scd' (Synchronicity 'cd') instead of 'cd' when using
DesignSync. See the help for 'scd' for a full comparison of these
commands.
```

SYNOPSIS

```
See a Tcl language reference manual.
```

SEE ALSO

```
scd, pwd, spwd
```

pwd

pwd Command

NAME

```
pwd                 - Displays the path of the current directory
```

DESCRIPTION

```
This command is the standard Tcl 'pwd' command. It displays the path
of the current working directory as viewed by the operating system.

In general, use the 'spwd' (Synchronicity 'pwd') command instead of
```

'pwd' when using DesignSync. See the help for 'spwd' for a full comparison of these commands. In most cases, however, the 'spwd' and 'pwd' commands differ only in that 'pwd' returns a path whereas 'spwd' returns a URL.

Refer to a Tcl language reference manual for a full description of the standard 'pwd' command.

SYNOPSIS

See a Tcl language reference manual.

SEE ALSO

spwd, cd, scd

scd

scd Command

NAME

scd - Changes your current location

DESCRIPTION

The 'scd' (Synchronicity 'cd') command lets you change your current location. Your location is typically a local folder, which corresponds to a file-system directory, but can be any container object. A container object is an object that contains other objects, such as a folder or a vault.

Specify a relative path, absolute path, or file: URL for local (client-side) folders, or a sync: URL for server-side container objects. Specifying 'scd' without an argument puts you in your home directory (as defined by $HOME on UNIX or your user profile, which is managed by the User Manager tool, on Windows platforms).

In dss/dssc, you can use the 'prompt' command to specify that your command-line prompt be the URL of your current location.

The standard Tcl 'cd' command and 'scd' differ in that 'cd' operates on file-system directories, whereas 'scd' operates on DesignSync container objects, such as folders and vaults. In the common case of local folders, these commands behave similarly because there is often a one-to-one correspondence between the

```
file-system directory and a DesignSync folder:
  dss> cd /home/tgoss/Projects/Sportster
  dss> pwd
  /home/tgoss/Projects/Sportster
  dss> spwd
  file:///home/tgoss/Projects/Sportster

  dss> scd /home/tgoss/Projects/Sportster
  dss> pwd
  /home/tgoss/Projects/Sportster
  dss> spwd
  file:///home/tgoss/Projects/Sportster

The behavior of 'scd' and 'cd' differ in the following cases:
  o You must use 'scd' when specifying server-side objects -- any
    object you identify with a sync: URL, such as vaults, branches,
    and server-side folders.
  o You cannot use 'scd' to navigate into Synchronicity client
    metadata directories (.SYNC). DesignSync does not recognize
    these directories as Synchronicity folders because you should not
    operate on these directories. However, you can use 'cd'
    because .SYNC directories are valid file-system directories.
      dss> scd .SYNC    <-- Fails
      The system cannot find the path specified:
         /home/tgoss/Projects/Sportster/code/.SYNC
      dss> cd .SYNC     <-- Succeeds
      dss> pwd
      /home/tgoss/Projects/Sportster/code/.SYNC
      dss> spwd
      file:///home/tgoss/Projects/Sportster/code
  o When used in server-side scripts, "cd /" puts you at the
    file-system root of the SyncServer host (/), whereas "scd /"
    puts you at the SyncServer's root directory as specified when
    the SyncServer was installed (for example,
    /usr1/Synchronicity/syncdata/gilmour/2647/server_vault).
```

SYNOPSIS

```
scd [--] [<path> | <URL>]
```

OPTIONS

- --

--

```
  --              Indicates that the command should stop
                  looking for command options. Use this option
                  when the path begins with a hyphen (-).
```

RETURN VALUE

```
none
```

SEE ALSO

```
spwd, cd, pwd, prompt
```

EXAMPLES


```
The following examples illustrate common uses of the 'scd'
command. In the first two examples, you could use 'cd' to
accomplish the same result. In the other examples, you must use 'scd'.
```

Example of Specifying an Absolute Path Name

```
The following example specifies an absolute path name and changes the
local folder to file:///home/userdir/test (DesignSync converts the
path to a URL).

  dss> scd /home/userdir/test
  dss> spwd
  file:///home/userdir/test
```

Example of Specifying a Relative Path Name

```
The following example specifies relative path names. For example, if
your current folder is /home/userdir and it contains the
subfolder 'test', either of the following commands changes to the
test folder:

  dss> scd ./test
  dss> scd test
```

Example of Specifying a Server-Side Vault Location

```
The following example changes your location to a vault
(server-side) folder:

  dss> scd sync://host3:2024/Projects/asic/test
  dss> spwd
  sync://host3:2024/Projects/asic/test
```

Example of Navigating on the Server

```
The following example shows that once you are located in a
server-side folder, you can use relative paths to navigate:

  dss> spwd
  sync://host3:2024/Projects/asic/test
  dss> scd ..
  dss> spwd
  sync://host3:2024/Projects/asic
  dss> scd code
  dss> spwd
  sync://host3:2024/Projects/asic/code
```

Example of Changing to a Calculated Server Directory

```
You can change location to a vault object (because it is a container
object that contains versions). You can either use a command that
computes the location (for example, the 'url vault' command)
or you can explicitly specify the URL:

  stcl> scd [url vault file5.v]
  stcl> spwd
  sync://host3/2024/Projects/asic/test/file5.v;

Note: If you specify the URL, you must precede the semicolon with a
backslash or surround the URL with double quotes to prevent
stcl from treating the semicolon as a command separator:

  stcl> scd sync://host3:2024/Projects/asic/test/file5.v\;
  stcl> spwd
  sync://host3/2024/Projects/asic/test/file5.v;
```

spwd

spwd Command

NAME

```
spwd                - Displays the URL of the current location
```

DESCRIPTION

```
The 'spwd' (Synchronicity 'pwd') command displays the URL of
the current location -- folder or other container object -- as
selected by the most recent 'scd' command. A container object is
any object that contains other objects, such as a folder or a vault.

The standard Tcl 'pwd' command and 'spwd' differ in that 'pwd'
operates on operating-system directories whereas 'spwd' operates on
Synchronicity container objects. Also, 'pwd' returns an absolute
path, whereas 'spwd' returns a URL. In most cases, there is a
one-to-one correspondence between the current working directory and
a Synchronicity folder, so you can use either command. For example:
  dss> scd /home/tgoss/Projects/Sportster
  dss> pwd
  /home/tgoss/Projects/Sportster
  dss> spwd
  file:///home/tgoss/Projects/Sportster

There are cases where 'spwd' and 'pwd' return different values:
  - If you use 'scd' to go to a server-side location, 'spwd'
    returns that location, whereas 'pwd' is unaffected (returns
    the location prior to the 'scd' command). This
    difference is because 'cd' and 'pwd' do not recognize
    server-side (sync:) objects.
       stcl> pwd
       /home/tgoss/Projects/Sportster
       stcl> spwd
       file:///home/tgoss/Projects/Sportster
       stcl> scd [url vault .]
       stcl> spwd
       sync://apollo:2647/Projects/Sportster
       stcl> pwd
       /home/tgoss/Projects/Sportster
  - If you use 'cd' to go to a metadata (.SYNC) directory, 'pwd'
    returns the path to the .SYNC directory, whereas 'spwd' is
    unaffected (returns the location prior to the 'cd' command).
    This difference is because DesignSync does not
    recognize .SYNC directories as folders so that users are
    discouraged from operating on them. For example:
       stcl> pwd
       /home/tgoss/Projects/Sportster/code
       stcl> spwd
       file:///home/tgoss/Projects/Sportster/code
       stcl> cd .SYNC
       stcl> pwd
       /home/tgoss/Projects/Sportster/code/.SYNC
       stcl> spwd
       file:///home/tgoss/Projects/Sportster/code
  - When used from a server-side script, 'pwd' returns paths
    relative to the file-system root directory (/) whereas 'spwd'
    returns URLs relative to the SyncServer's root directory.
```

SYNOPSIS

```
spwd
```

OPTIONS

```
none
```

RETURN VALUE

```
Returns the URL of the current location.
```

SEE ALSO

```
scd, pwd, cd
```

EXAMPLES

Example of Using spwd on a Local Folder

```
This example shows the return value from the 'spwd' command
when the previous 'scd' command selected a local folder. In this
common case, 'pwd' returns the same value, except as a path instead
of a URL.

 dss> scd /home/syncmgr/test
 dss> spwd
 file:///home/syncmgr/test
 dss> pwd
 /home/syncmgr/test
```

Example of Using spwd on a Vault Folder

```
This example shows the return value from 'spwd' when the previous
'scd' selected a vault folder.

 stcl> scd [url vault test_dir]
 stcl> spwd
 sync://host3:2024/Projects/ASIC/test_dir
```

Example of Using spwd on a Vault File Object

```
This example shows the result of 'spwd' when the previous 'scd'
selected a vault object.

 stcl> scd [url vault myfile.txt]
 stcl> spwd
 sync://host3:2024/Projects/ASIC/test_dir/myfile.txt;
```

Informational

annotate

annotate Command

NAME

```
annotate            - Shows last modification information per line
```

DESCRIPTION

```
This command opens the selected text file object and displays last
modification information.  The last modification information tells
you:
 o The last-modified version for the line.
 o The author credited with the changes
 o The date the modification was checked in.

The annotate command supports the command line default system.
```

SYNOPSIS

```
annotate [-back <number> | -from <selector>] [-output <filename> ]
[-version <selector>] [-[no]white] [--] <argument>
```

ARGUMENTS

- Workspace File
- Server File

Workspace File

```
<workspace file>  Displays the specified file version loaded in the
                  workspace. You may specify the file as either an
```

absolute or relative path. Because this command only supports a single argument, You may not use wildcards, even if the wildcard selection results in only a single file being identified.

Server File

<server file> Displays the specified file version. Specify the object with the sync URL in the format: sync://<host>:<port>/<path>/<object>;<selector>

OPTIONS

- -back
- -from
- -output
- -version
- -[no]white
- --

-back

-back <number> Specifies the number of versions to consider when creating the annotated document. The versions included in the annotation begin with the specified version (-version option, if selected) and each version is processed until the specified number of versions back is reached, then the annotated file is generated.

If neither the -back nor the -from option is specified, the annotate includes the entire object history, beginning with the vault root. (Default)

Note: -back is mutually exclusive with -from.

-from

-from <selector> Specifies the selector of the first version to consider when created the annotated document. The versions included in the annotation begin with the specified version (-version) and end with the version that resolves to the selector specified with the -from option. The specified selector must resolve to a version on a path from the annotated version to the vault root.

Note: For module member version, the selector must be the module member version number.

If neither the -back nor the -from option is specified, the annotate includes the entire object history, beginning with the vault root. (Default)

Note: -from is mutually exclusive with -back.

-output

-output <file> — Sends the results of the annotate command to the named file. The contents can then be processed or viewed as needed.

-version

-version <selector> — Specifies the version of a file to display. If no version is specified, DesignSync uses the version loaded in the workspace. (Default)

You may specify any valid single selector. Note: When you use a version number to specify a module member, use the module version of the module containing the module member version you're interested in.

-[no]white

-[no]white — Specifies whether to ignore leading and trailing whitespace changes.

-nowhite indicates the whitespace changes are considered a modification. Therefore if the indentation level was changed, the line is considered modified. (Default)

-white indicates the whitespaces changes are not considered a modification. For example, if a user changes the indent level, the line is not considered modified. The last textual change (or embedded whitespace change) made is considered the last modification.

--

-- — Indicates that the command should stop looking for command options. Use this option when arguments to

```
                the command begin with a hyphen (-).
```

RETURN VALUE

```
If the annotate command is successful, DesignSync returns an
empty string (""). If the command cannot run, DesignSync throws an
error message explaining the failure.
```

SEE ALSO

```
ls, vhistory, selectors
```

EXAMPLES

```
This example shows the annotate command with a fragment of the
collection.ctp script included in the sample directory:

dss> annotate collection.ctp
  Beginning Annotate operation...
  ...
 1.1          (barb    9-Apr-06): # Get the base name of a file.
 1.1          (barb    9-Apr-06): proc collectionCTP::getBase
  {filename} {
 1.2.1.3      (ian     1-Mar-07):     set tail [collectionCTP::tail
  $filename]
 1.2.1.2      (ian     1-Feb-07):     set dot [string first
  . [collectionCTP::tail $filename]]
 1.1          (barb    9-Apr-06):     if {$dot == -1} {
 1.1          (barb    9-Apr-06):         return $filename
 1.1          (barb    9-Apr-06):     }
 1.2.1.3      (ian     1-Mar-07):     set bit [expr [string length
  $filename] - [string length $tail] + $dot - 1]
 1.2.1.3      (ian     1-Mar-07):     return [string range $filename
  0 $bit]
 1.1          (barb    9-Apr-06): }
```

compare

compare Command

NAME

```
compare             - Compares two defined sets of files or objects
```

DESCRIPTION


```
The 'compare' command allows you to compare two versions of a module,
two legacy configurations in a vault, or to compare workspaces to
modules, vaults, legacy configurations or other workspaces.

Note: The compare command compares only collections and not
collection members. The compare command doesn't compare empty
directories.

The compare command has a number of standard arguments, and then a
specification of what can be compared, in the form of either 0, 1 or
2 arguments, plus 0, 1, or 2 selectors. The arguments can be any
directory path or module instance. For more information on arguments,
see the ARGUMENTS section. The selectors can be any valid selector or
selector list, except that they may NOT contain the Date() or
VaultDate() items. For more information on selectors, see the
selectors topic.

Note: The compare command works from the path of the object, not from
the UUID, this means that if an object has moved, it may be reported
twice, one in the original location and once in the new location.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

This command supports the command defaults system.
```

Understanding the Types of Possible Compare Operations

```
The following table describes the action of the compare command
when you specify one or more selectors and one more arguments.

The selectors can be any valid selector or selector list, except that
they may NOT contain the Date() or VaultDate() items. For more
information on selectors, see the selectors topic.

The arguments can be any directory path or module instance. For more
information on arguments, see the ARGUMENTS section.

Any combination not indicated is disallowed.
```

Notes:
o This command is not intended to provide a way to compare two different modules, so there is no way to specify two module URLs.

o For space considerations, the values of selector1/selector2 and the arguments allowed are represented as sel1/sel2 and arg1/arg2.

sel	sel2	arg1	arg2	Description
No	No	Yes	No	Compare the contents of the specified argument against the server version.

sel1	sel2	arg1	arg2	Description
No	No	No	No	Compare the current workspace directory path against the associated server version.

sel1	sel2	arg1	arg2	Description
No	No	Yes	Yes	Compare the two arguments.

sel1	sel2	arg1	arg2	Description
Yes	No	Yes	No	Compare the specified argument against the version indicated by the specified selector. The selector value is always evaluated against the server version of the argument. If you've filtered data out of your workspace and do not use the corresponding filters on the compare command, you see that data listed as present on the server, but not in your workspace.

sel1	sel2	arg1	arg2	Description
Yes	No	No	No	Compare the current workspace against the specified selector. This uses the server version that corresponds to the persistent selector set on the workspace, rather than the current workspace module version. The selector value is always evaluated against the server version of the module or DesignSync vault. If you've filtered data out of your workspace and do not use the corresponding filters on the compare command, you see that data listed as present on the server, but not in your workspace.

sel1	sel2	arg1	arg2	Description
Yes	Yes	No	No	Compare the two specified versions. Both are server versions identified by selector, for example you might compare Rel1:Beta against Re1:Gold, or Rel2:Beta. Neither of these is required to be populated into a workspace on your system in order to do the comparison.

sel1	sel2	arg1	arg2	Description
Yes	Yes	Yes	No	Compare the two specified selector versions for the argument given. If the argument is a workspace path, the command uses the vault associated with the workspace path.

Understanding the Output

The output can be formatted for easy viewing (-format text) or optimized for Tcl processing (-format list). Both viewing formats show the same information, but may have different names. In the table below, the Column Titles column shows the text output column header and the Property Names column shows list output key value.

This information is returned by the compare command regardless of what report mode you specify. Different report modes add additional information as described in the Options section under -report.

Understanding Status Values in the Ouput

The following table describes the status values:

Column Titles	Property Names	Description
Identical	identical	The objects are the same.
Different versions	different_versions	The objects are the same but are at different versions.
Different objects	different_objects	The objects are the same natural path and the same versions, but they do not have the same unique ID values.
First only	first_only	The object is present in the first area only.
Second only	second_only	The object is present in the second area only.
Different states	different_states	The objects are the same version, but in different states, for example one item is modified or absent (in reference mode) while the other is not.
modified	modified	The objects are the same (same version and same uids), but both are modified and therefore the files might be different.

Content identical	identical_content	The objects are the same (same version, same uid, same checksum), but the versions are different. Note: If your workspace is populated in share, reference, or mirror mode, DesignSync does not retain checksum information and these workspaces will never register as Content identical.

Running Compare on Modules (Module-based)

You can run the compare command to:

- Show all the files that were changed, added, or removed between module versions. If you have a version tagged for release and want to compare it against the module manifest of a previous or follow-on versions, you can use this to determine what files have changed.

- Compare two workspaces. If you are running a simulation in your workspace, and a coworker is running simulations in his workspace, but you are seeing different results, you can compare your workspaces to see what is different.

- Report items that are the same and history information when different. Also, in order to understand what the changes are, you can see the checkin comment history from the different versions back to their common ancestor.

- Produce output for further processing. For example, having compared your workspace with that of someone else, you would like to take the list of what is different and tag those items.

- Report hierarchies in only one side of a comparison. If you are comparing two hierarchies, but only some of the sub-directories are present in one of the hierarchies, you can view the full contents of those directories, or just get a note that the directory is present in only one of the areas.

Note: When compare includes a workspace that has been populated with a selector list, creating a blended workspace, the objects in the workspace are compared against both the module indicated by the main selector and other selectors. Therefore a member populated from the main selector is compared against the corresponding vault version and a member populated from the selected list blended into the workspace is compared against that module version.

The hrefmode options respect the traversal method identified by the "HrefModeChangeWithTopStaticSelector" registry key. For more information, see the "Module Hierarchy" topic in the ENOVIA Synchronicity DesignSync Data Manager User's Guide.

Note: If a workspace has been populated with overriding hrefs, the compare command uses the overridden submodules as part of the comparison operation.

Understanding Columns Returned When Comparing Module Objects (Module-based)

Note: The column title for the path properties may change depending on whether you are comparing workspaces (Workspace Version) or legacy modules (Configuration Version).

Column Titles	Property Names	Description
Workspace/ Configuration Version	path1	The path of the first argument specified by the command.
Workspace/ Configuration Version	path2	The path of the second argument specified by the command.
Status	state	The status value shows the result of the comparison. The next table shows all the status values possible. Note: The list view shows the overall status (state) of the files, and the specific information about both versions being compared.
Name	name	Name of the object being compared.
	type	Type of object being compared. Type values include: o file o module o folder o project
	url	The URL of the module. (module type only)
	version	Version of the module. (module type only)
	relpath	The relative path to that module from the top level module.
	modulepath	When only a portion of a module is being reported, for example, a single directory is specified within a module, the command compares only the contents of the sub-modules that fall under that directory. If a sub-module has no contents under that directory then the sub-module is contained in the results but has no listed objects. The sub-module may also have a

modulepath property which indicates the path within that module for which the data is included.

Note: Module members that have been moved appear twice in the compare output, once in their original location and once in their new location with "First only" or "Second only" status values.

Using Compare with Legacy Module Objects (Legacy-based)

You can run the compare command to:

- Compare two workspaces. If you are running a simulation in your workspace, and a coworker is running simulations in his workspace, but you are seeing different results, you can compare your workspaces to see what is different.

- Report items that are the same and history information when different. Also, in order to understand what the changes are, you can see the checkin comment history from the different versions back to their common ancestor.

- Produce output for further processing. For example, having compared your workspace with that of someone else, you would like to take the list of what is different and tag those items.

- Compare a legacy module configuration in your workspace to a released legacy module configuration.

Note: There is no means to compare a workspace against a default legacy module configuration other than explicitly specifying the configuration as Trunk.

Using Compare with File-Based Objects (File-based)

You can run the compare command to:

- Compare two workspaces. If you are running a simulation in your workspace, and a co-worker is running simulations in his workspace, but you are seeing different results, you can compare your workspaces to see what is different.

- Report items that are the same and history information when different. Also, in order to understand what the changes are, you can see the checkin comment history from the different versions back to their common ancestor.

- Produce output for further processing. For example, having compared your workspace with that of someone else, you would like to take the list of what is different and tag those items.

- Compare the workspace version to a server version. For example,

you can see differences in your local workspace, or changes from the original or other versions.

- Compare two server versions. For example, you can compare the a version on one branch or a version on another branch, or compare two tagged released versions against each other to see how they differ.

Understanding Columns Returned When Comparing File Objects (File-based)

Note: The column title for the path properties may change depending on whether you are comparing workspaces (Workspace Version) or legacy modules (Configuration Version).

Column Titles	Property Names	Description
Workspace/ Configuration Version	path1	The path of the first argument specified by the command.
Workspace/ Configuration Version	path2	The path of the second argument specified by the command.
Status	state	The status value shows the result of the comparison. The next table shows all the status values possible. Note: The list view shows the overall status (state) of the files, and the specific information about both versions being compared.
Name	name	Name of the object being compared.
	type	Type of object being compared. Type values include: o file o module o folder o project

SYNOPSIS

```
compare [-format list|text] [-[no]history] [-exclude <string>]
        [-filter <string>] [-hreffilter <string>]
        [ -modulecontext <context>] [-output <file> |
          -stream <stream>]  [-[no]path] [-recursive | -norecursive]
        [-report silent|brief|normal|verbose] [-[no]same]
        [-selector <selector> [-hrefmode dynamic|static|normal]]
        [-selector2 <selector2> [-hrefmode2 dynamic|static|normal]]]
        [-view <viewName>[,<ViewName>[,...]]] [--] [argument [argument]]
```

ARGUMENTS

Module Folder (Module-based)

```
<module folder>    Compares the contents of the specified module
                   workspace or server directory. If the workspace
                   directory contains more than one module, you can
                   restrict the compare to a single module by using
                   the -modulecontext option.

                   If a module folder is specified, the
                   -modulecontext option is required.
```

DesignSync Folder (File-based)

```
<DesignSync folder> Compares the contents of the specified folder,
                    and, when used with the recursive option, all
                    subfolders.
```

Server Folder (File-based)

```
<server folder>    Compares the contents of the specified folder
                   on the server, and when used with the -recursive
                   option, all subfolders. Specify the object with
                   the sync URL in the format:
                   sync://<host>:<port>/<path>/<folder>
```

OPTIONS

- -report
- -[no]same
- -selector (Module-based)
- -selector (File-based)
- -selector2 (Module-based)
- -selector2 (File-based)
- -view (Module-based)
- --

-exclude

```
-exclude <expr>    Excludes items that match the given regular
                   expression.

                   The expression is matched against the object
                   path that would be reported. If '-path' is
                   specified, the command matches the expression
                   against the relative path; if '-fullpath' is
                   specified, the command matches the expression
                   against the full path, else against the object
                   leaf name.

                   By default, the 'compare' command does not
                   exclude the objects in the global exclude lists
                   (set using Tools->Options->General->Exclude
                   Lists or using SyncAdmin:General->Exclude Lists).
                   To exclude these objects from a 'compare' listing,
                   apply the -exclude option with a null string:
                      compare -exclude ""
                   The objects in the global exclude lists are
                   appended to the 'compare' exclude list if
                   you exclude other values:
                      compare -exclude "README.txt"
```

-filter (Module-based)

```
-filter <string>   Specify one or more extended glob-style
                   expressions to identify an exact subset of
                   module objects on which to operate. Use the
                   -exclude option to filter out DesignSync
                   objects that are not module objects.

                   The -filter option takes a list of expressions
                   separated by commas, for example: -filter
                   +top*/.../*.v,-.../a*

                   Prepend a '-' character to a glob-style expression
                   to identify objects to be excluded (the default).
                   Prepend a '+' character to a glob-style expression
                   to identify objects to be included.  Note that if
                   the list of expressions begins with an include
```

character ('+'), the filter excludes all objects except those that match the include string.

Specify the paths in your glob-style expressions relative to the current directory, because DesignSync matches your expressions relative to that directory. For submodules followed through hrefs, DesignSync matches your expressions against the objects' natural paths, their full relative paths. For example, if a module, Chip, references a submodule, CPU, and CPU contains a file, '/libs/cpu/cdsinfo.tag', DesignSync matches against '/libs/cpu/cdsinfo.tag', rather than matching directly within the 'cpu' directory.

If your design contains symbolic links that are under revision control, DesignSync matches against the source path of the link rather than the dereferenced path. For example, if a symbolic link exists from 'tmp.txt' to 'tmp2.txt', DesignSync matches against 'tmp.txt'. Similarly for hierarchical operations, DesignSync matches against the unresolved path. If, for example, a symbolic link exists from dirA to dirB, and dirB contains 'tmp.txt', DesignSync matches against 'dirA/tmp.txt'.

The extended glob-style expressions you use to filter the objects are standard glob-style expressions, but they are extended so that you can use the "..." syntax to indicate that the expression matches any number of directory levels. For example, the expression, "top/.../lib/*.v" matches *.v files in a directory path that begin with "top", followed by zero or more levels, with one of those levels containing a lib directory. The command traverses the directory structure. If a directory name matches an exclude clause of the filter, then the entire directory and all its contents are filtered (the command stops descending at that point), otherwise the command continues traversing the directory structure searching for matching objects.

The -filter option does not override the exclude list set using SyncAdmin's General=>Exclude Lists tab or with the -exclude command line option; the items in the exclude list are combined with the filter expression. For example, an exclude list of "*%,*.reg" combined with '-filter .../*.doc' is equivalent to: '-filter .../*.doc,.../*%,.../*.reg'.

-format

```
-format list|text  Determines the format of the output.
                   Valid values are:
                   o text   Display a text table with headers and
                            columns. (Default) Objects are shown in
                            alphabetical order.

                   o list   Tcl list structure, designed for further
                            processing, and for easy conversion to a
                            Tcl array structure. This means that it
                            is a list structure in name-value pair
                            format. The top level structure is:
                            {
                               path1 <path>
                               path2 <path>
                               type folder
                               objects <object_list>
                            }

                            path1 and path2 are the areas compared,
                            and may be local workspace paths, or
                            configuration URLs in the form:
                            sync://machine:port/path@config

                            The "selector" part is the originally
                            supplied selector name, rather than
                            any name resulting from selector
                            expansion.

                            object_list is then defined as a list of
                            items of the form:
                            {
                               name <object name>
                               type1 folder | file
                               type2 folder | file
                               objects <object_list>
                               state <status>
                               props1 <prop_list>
                               props2 <prop_list>
                            }

                            The "name" is the name of the object, and
                            may contain the relative path from the
                            starting point if the '-path' argument
                            was specified.

                            Note: When comparing a module hierarchy,
                            the top level list includes a
                            "modules" property whose value is a
                            list of the results for each module in
                            the hierarchy.

                            The "type1" and "type2" properties
                            indicate whether the object is a folder,
                            file, or module. Note that collection
                            objects have a type file, also all
                            symbolic links, whether to files or
```

directories, have a type of "file". The reason that all links are "file" is to be consistent with the "ls" command which treats links as files.

The type may be different on the two sides. A folder or module have object lists; which a file object does not have. A file has props1 and props2 lists which folders and modules do not have.

The state value contains the status of the object. For more information on status values, see the Status table in the "Understand the Output" section above.

The props1 and props2 are lists of properties for the object from path1 and path2, in the form:

```
{
   version <version>
   state <state>
   ancestor <version>
   history <history_list>
}
```

To process the results, use the "compare-foreach" function below.

-[no]history (File-based)

-[no]history	Determines whether the command returns the version history of the version being compared from the common ancestor of the two versions. -nohistory does not return version history information. (Default) -history returns the checkin comment and other details for the version history back to the common ancestor of the two versions being compared.

-hreffilter (Module-based)

-hreffilter <string>	Excludes href values during recursive operations on module hierarchies. Because hrefs link to submodules, you use -hreffilter to exclude particular submodules. The hreffilter value is matched against both the name of the href and the target module name. Note that unlike the -filter option which lets you include and exclude

items, the -hreffilter option only excludes hrefs and, thus, their corresponding submodules.

Specify the -hreffilter string as a glob-style expression. The string must represent a simple leaf name; you cannot specify a path. DesignSync matches the specified href filter against hrefs anywhere in the hierarchy. Thus, DesignSync excludes all hrefs by this leaf name; you cannot exclude a unique instance of the href.

You can prepend the '-' exclude character to your string, but it is not required. Because the -hreffilter option only supports excluding hrefs, a '+' character is interpreted as part of the glob expression.

-hrefmode (Module-based)

-hrefmode dynamic|static|normal	Specifies how the hierarchy is processed when -selector is specified. o dynamic - Expands hrefs at the time of the operation to identify the version of the submodules being compared. o static - Expands with the submodules versions referenced by the hrefs when the module version was initially created. o normal - Expands the hrefs at the time of the compare operation until it reaches a static selector. If the reference uses a static version, the hrefmode is set to 'static' for the next level of submodules to be compared; otherwise, the hrefmode remains 'normal' for the next level. (Default). This behavior can be changed using the "HrefModeChangeWithTopStaticSelector" registry key to determine how hrefs are followed. Note: Specifying different -hrefmodes with the same value for selector and selector2 allows comparison of the different resulting hierarchies.

-hrefmode2 (Module-based)

-hrefmode2 dynamic|static normal	Specifies how the hierarchy is processed when -selector2 is specified. o dynamic - Expands hrefs at the time of the operation to identify the version of the submodules being compared. o static - Expands with the submodules versions referenced by the hrefs when the module version was initially created.

o normal - Expands the hrefs at the time of the compare operation until it reaches a static selector. If the reference uses a static version, the hrefmode is set to 'static' for the next level of submodules to be compared; otherwise, the hrefmode remains 'normal' for the next level. (Default). This behavior can be changed using the "HrefModeChangeWithTopStaticSelector" registry key to determine how hrefs are followed.

Note: Specifying different -hrefmodes with the same value for selector and selector2 allows comparison of the different resulting hierarchies.

-modulecontext (Module-based)

-modulecontext <context>	Identifies the module being compared. The -modulecontext option restricts the compare to only a particular module if your workspace has overlapping modules so that you can indicate which module you want to compare. Specify the desired module using the module name (for example, Chip), or a module instance name (for example, Chip%0), or server module URL (sync://server1:2647/Modules/Chip). If you use module context to specify a server object, you must specify the latest version. Note that you cannot use a -modulecontext option to operate on objects from more than one module; the -modulecontext option takes only one argument, and you can use the -modulecontext option only once on a command line.

-output

-output <file> | -stream <stream>	Outputs the result to the specified file or stream. The output file is used to preserve the results for later viewing or distribution. If the specified file already exists, it is overwritten with the new information. The stream option passes the results to named Tcl stream. Depending on whether you open the stream using the Tcl 'open' command in write (w) or append (a) mode, you can overwrite or append to an existing file.

Note: The -stream option is only applicable in the stcl and stclc Tcl shells, not in the dss and dssc shells.

If neither -output nor -stream is specified, the command output is displayed on the screen.

-[no]path

-[no]path

Controls the format of the path that is reported for each object. Objects are reported on a per-directory basis, with each directory path given as a full URL. The items within the directory can be reported as:

-nopath displays simple object names with no directory path. (Default)

-path display a relative path to the start of the command.

Notes:
o If the -report silent option is specified, the -path option is automatically used.

o The 'contents' option to report as a full url (-fullpath) is not supported by this command, because each object will potentially have two full URLs for the two areas being compared.

-[no]recursive (Module-based)

-[no]recursive

Determines whether the compare command operate on the specified argument or all subfolders in the the argument's hierarchy, or all submodules in the argument's hierarchy.

-recursive performs this operation on all subfolders in the hierarchy, or on all sub-modules in a module hierarchy when the arguments are modules or modulecontext is used.

-norecursive performs this operation on the specified folder only. (Default)

Note: To filter modules, use the -hreffilter option. If the folder contains multiple modules, you can restrict your compare to a single module by using the -modulecontext option.

-[no]recursive (Legacy-based)

-[no]recursive

Determines whether the compare command operate on the specified argument or all subfolders in the the argument's hierarchy, or all submodules in the argument's hierarchy.

-recursive performs this operation on all subfolders in the hierarchy, or on all sub-modules in a module hierarchy when the arguments are modules or modulecontext is used.

-norecursive performs this operation on the specified folder only. (Default)

Note: The -nomodulerecursive option has been deprecated. For legacy modules, use the -norecursive option.

-[no]recursive (File-based)

-[no]recursive

Determines whether the compare command operate on the specified argument or all subfolders in the the argument's hierarchy.

-recursive performs this operation on all subfolders in the hierarchy.

-norecursive performs this operation on the specified folder only. (Default)

-report

-report

Controls the level of additional information reported as the command progresses.

- silent — Returns only the primary return data for the command - the data that has been compared, and whether the data is the same.

  Note: Using format -text, the relative path is returned in silent mode, unless the -fullpath option is specified.

- brief — Includes header lines showing what was compared and status lines where a long command might be performed without any output, such as when gathering data from a remote server. Directories that contain only items on one side, or for which all items are identical on both sides are not expanded to show their contents.

o normal Expands directories that would be skipped in brief output mode, because they are present in one area only, or because all items are identical on both sides. (Default)

o verbose Includes information on configuration mappings.

In the output from brief mode, a directory may be shown with the message "(Nothing / all identical on this side)". This message indicates either that all items under this folder are the same on both sides or that there are items to report only on this side of the comparison.

Note: The version is shown as 'Unknown' if the version of the file in the workspace cannot be determined from the local metadata. If an object has no local metadata, its Version will be 'Unmanaged'. Recreated files will appear as 'Unmanaged', because they have no local metadata (their metadata was removed by a previous 'rmfile').

-[no]same

-[no]same Determines whether the output includes only items that are different or items that are the same and items that are different.
-nosame reports only items that are different. (Default)
-same reports items that are the same, in addition to items that are different.

-selector (Module-based)

-selector <selector> Specifies the selector to compare. When only one selector option is used, the object specified by the selector is compared to a workspace. You cannot use a Date() or VaultDate() selector.

Note: When specifying a selector, you must distinguish between branch and version selectors. If you are specifying a branch and the branch is anything other than Trunk, specify both the branch and version as follows:
'<branchtag>:<versiontag>', for example, 'Rel2:Latest'. You can also use the shortcut, '<branchtag>:', for example, 'compare -selector B1:' or 'compare -selector B1:gold'.

This option can not take a selector list. You must specify a single selector for each selector option.

-selector (File-based)

-selector <selector>	Specifies the selector to compare. When only one selector option is used, the object specified by the selector is compared to a workspace. You cannot use a Date() or VaultDate() selector. Note: When specifying a selector, you must distinguish between branch and version selectors. If you are specifying a branch and the branch is anything other than Trunk, specify both the branch and version as follows: '<branchtag>:<versiontag>', for example, 'Rel2:Latest'. You can also use the shortcut, '<branchtag>:', for example, 'compare -selector B1:' or 'compare -selector B1:gold'.

-selector2 (Module-based)

-selector2 <selector2>	Specifies a second selector when comparing two modules, legacy module configurations, or directories in the vault. You cannot use a Date() or VaultDate() selector. Note: When specifying a selector, you must distinguish between branch and version selectors. If you are specifying a branch and the branch is anything other than Trunk, specify both the branch and version as follows: '<branchtag>:<versiontag>', for example, 'Rel2:Latest'. You can also use the shortcut, '<branchtag>:', for example, 'compare -selector B1:' or 'compare -selector B1:gold'. This option can not take a selector list. You must specify a single selector for each selector option.

-selector2 (File-based)

-selector2 <selector2>	Specifies a second selector when comparing two modules, legacy module configurations, or directories in the vault. You cannot use a Date() or VaultDate() selector. Note: When specifying a selector, you must

distinguish between branch and version selectors. If you are specifying a branch and the branch is anything other than Trunk, specify both the branch and version as follows: '<branchtag>:<versiontag>', for example, 'Rel2:Latest'. You can also use the shortcut, '<branchtag>:', for example, 'compare -selector B1:' or 'compare -selector B1:gold'.

-view (Module-based)

-view <viewName> [,<viewName>[,...]]

Specifies a view name or list of view names to use when comparing the module. The view list is a comma-separated list of view names.

If you compare a workspace module and a server module, the view refers to the sever module. The workspace contents are the objects loaded in the workspace.

Note: The -view option requires a specified -selector option.

--

--

The command option '--' indicates that following arguments should not be taken as options, but as paths that begin with a '-'.

RETURN VALUE

Empty string if -format value is text.
Tcl list if the -format value is list.
Empty string if -output or -stream is used with -format.

When run from a server-side script, the server-side URL used in the results is relative to the server root (meaning the host:port information is omitted), for example: "sync:///Projects/p1."

SEE ALSO

compare-foreach, contents, contents-foreach, command defaults

EXAMPLES

Example of Comparing Two Selectors

```
Compare the two given selectors, using the 'url vault' of the current
working directory to identify the server to work from.

  dss> compare -selector relA -selector2 relB -recursive
```

Example of Comparing Two Selectors with a URL

```
Compare the two given selectors, starting from the given URL.

  dss> compare -selector relA -selector2 relB \
       sync://saturn.ABCo.com:30003/Modules/df2test -recursive
```

Example of Comparing the Current Directory Against Another Directory

```
Compare my current directory against the other one given. Compare
only this directory and not the sub-directory contents.

  dss> compare . /home/users/fred/Modules/P1 -norecursive

  (Note: You are not required to specify the -norecursive option;
  the behavior of the compare command is nonrecursive by default.)
```

Example of how to use '-format list' option

```
To find which objects in your workspace have or do not have a
specific tag, use 'compare' to compare the workspace with the tag
configuration. Then report the items that do or do not match.

 Create an auto-loaded Tcl proc:

 proc has_tag {dir tag {no_tag 0}} {
```

```
    record {set cres [compare -selector $tag $dir -same -recursive \
            -format list -path -report verbose]} nolog

    compare-foreach obj1 obj2 $cres {
      if {[info exists obj1(version)]} {
          if {[info exists obj2(version)] && \
               ([string compare $obj1(version) $obj2(version)] == 0)} {
              if {!$no_tag} {
                  puts $obj1(name)
              }
          } else {
              if {$no_tag} {
                   puts $obj1(name)
              }
          }
      }
    }

}
```

Note that this will not include unmanaged objects. To report unmanaged files, modify the code to add an "else" clause to the first "if" statement.

The code can be changed to return a list of the objects by changing the "puts" statements to commands to build a return list. See standard Tcl programming documentation for how to do that.

See the ENOVIA Synchronicity stcl Programmer's Guide for details on auto-loading.

To report which files in the workspace have a "baseline" tag:

```
stcl> has_tag . baseline
code/samp.s19
code/samp.lst
stcl>
```

To report which files in the workspace do not have a "baseline" tag:

```
stcl> has_tag . baseline 1
code/samp.asm
code/test.mem
code/sample1.asm
code/samp.mem
code/test.asm
top/alu/alu.v
stcl>
```

Example Comparing a Workspace to a Server Module Version (Module-based)

```
Comparing a workspace to a module.
stcl> compare -recursive -report brief -selector Gold -modulecontext \
CPU%0 /home/rsmith/MyModules/cpu
```

```
  Gathering data from vault sync://srv2.ABCo.com:2647/Modules/CPU@Gold
  Gathering workspace contents for module
  /home/rsmith/MyModules/cpu/CPU%0 within module path
  /home/rsmith/MyModules/cpu
  Comparison of (identical objects not reported):
  Workspace: /home/rsmith/MyModules/cpu/CPU%0
  Configuration: sync://srv2.ABCo.com:2647/Modules/CPU@Gold

  Workspace         Configuration          Status             Object
  Version           Version                                   Name
  1.2               1.1                    Different versions cpu.doc

  Comparison of (identical objects not reported):
  Workspace: /home/rsmith/MyModules/cpu/ALU/ALU%1
  Configuration: sync://srv2.ABCo.com:2647/Modules/ALU@1.2

 Note: For two workspaces, the headings will be "Workspace1: " and
 "Workspace2: " and the column titles "Workspace1 Version" and
 "Workspace2 Version". Similarly, when comparing two modules or
 legacy modules, they will be "Configuration1" and "Configuration2".
```

Example of Compare the Current Workspace Against A Module (Module-based)

```
Compare the current workspace against the module identified by the
given selector list.

  dss> compare -selector relA,relB -recursive
```

Example of Current Workspace Against Server Module Version (Module-based)

```
Compare the current workspace against the specified target module,
the latest versions on the Beta branch.  Notice that you append
':Latest' to the selector to indicate that Beta is a branch name and
not a version name.

  dss> compare -recursive -selector \
       sync://saturn.ABCo.com:30003/Modules/df2test@Beta:Latest
```

Example of Comparing a Module with different Hrefmodes (Module-based)

```
Compare a module to the same module using different hrefmodes.
  dss> compare -selector relA -selector2 relB \
       -hrefmode dynamic -hrefmode2 static -recursive
```

Example of Comparing a Tagging Module Version Against Latest (Module-based)

```
Compare tagged version of a specified module against the latest
version on the specified date.
```

```
dss> compare -selector Gold -selector2 Trunk:Date(01/31/09)
  sync://srv2.ABCo.com:2647/Modules/ChipDesigns/Chip

   Comparison of (identical objects not reported):
   Configuration1:
   sync://srv2.ABCo.com:2647/Modules/ChipDesigns/Chip@Gold
   Configuration2:
   sync://srv2.ABCo.com:2647/Modules/ChipDesigns/Chip@Trunk:Date(01/31/09)

  Configuration1         Configuration2         Status             Object
  Version                Version                                   Name
  1.3                    1.2                    Different versions chip.c
```

Example of Comparing the Workspace Version to the Server Version (File-based)

```
This example shows comparing the workspace version to the server
version.

  dss> compare /home/users/jsmith/workspace/proj1 -recursive
```

Example of Comparing Two Workspace Directories (File-based)

```
Compare the two given workspace directories. Note that to compare
your current workspace against someone else's, you must specify both
directories.

  dss> compare /home/users/fred/workspace/proj1 . -recursive
```

compare-foreach

compare-foreach Command

NAME

```
compare-foreach     - Function to process the results of a compare
                      command
```

DESCRIPTION

```
This routine loops over the items in a "compare" results list, and
processes each item in turn.
```

SYNOPSIS

```
compare-foreach <var1> <var2> <result_list> <tcl_script> [-nofolder]
                [-path]
```

ARGUMENTS

- Loop Variables
- Result List
- Tcl Script

Loop Variables

var1, var2	These are the loop variables. They are treated as Tcl arrays, and on each loop around contain the set of properties for the next object in the result_list, with var1 containing the "props1" properties and var2 the "props2" properties. In addition to the properties in the "props1/2" values for each object, the arrays will contain a "name" property and a "type" property, which are the name and type properties for the object.

Result List

result_list	This is the list of objects to be processed. It must be the result value of a call to the "compare" command with the "-format list" option.

Tcl Script

tcl_script	This is the piece of Tcl code that is executed on each loop.

OPTIONS

- -nofolder
- -path (Module-based)
- -path (File-based)

-nofolder

-nofolder	If specified, then the tcl_script will not be called for Folder type objects in the result_list.

-path (Module-based)

```
-path           The "name" property on each loop is usually just the
                "name" property for the object. However, if this
                option is specified, and a recursive "compare" was
                performed, then the "name" property is the relative
                path to each object. Normally, you would run
                "compare" with the -path option, in which case the
                "name" property contains an appropriate relative
                path. If you did not do that, then passing the
                "-path" option to compare-foreach will mean that the
                "name" property contains the relative path for each
                item, thus allowing you to differentiate between
                items with the same name in different folders.

                Note: For sub-modules, the relative path is always
                relative to the base directory of that module. To
                find the full relative path to an object from the top
                module, the path to the object needs to be prepended
                with the relative path to that module.
```

-path (File-based)

```
-path           The "name" property on each loop is usually just the
                "name" property for the object. However, if this
                option is specified, and a recursive "compare" was
                performed, then the "name" property is the relative
                path to each object. Normally, you would run
                "compare" with the -path option, in which case the
                "name" property contains an appropriate relative
                path. If you did not do that, then passing the
                "-path" option to compare-foreach will mean that the
                "name" property contains the relative path for each
                item, thus allowing you to differentiate between
                items with the same name in different folders.
```

SEE ALSO

```
compare
```

EXAMPLE

- Example of Using compare-foreach On a Result List From compare

Example of Using compare-foreach On a Result List From compare

```
Example of using the compare-foreach to parse a compare.

set result_list [compare -selector RelA -selector2 RelB -rec -format list]
```

```
compare-foreach obj1 obj2 $result_list {
    puts "Object: $obj1(name), state1: $obj1(state), state2: $obj2(state)"
}
```

contents

contents Command

NAME

```
contents            - Lists the contents of a configuration or a
                      module
```

DESCRIPTION

- Using Contents on Modules (Module-based)
- Understanding Module Hierarchy Output (Module-based)
- Understanding the path option (Module-based)
- Using Contents on Legacy Modules (Legacy-based)
- Notes for legacy modules (Legacy-based)
- Using Contents on File-Based Objects (File-based)

```
The 'contents' command provides a simple way to list the contents
of a DesignSync configuration and the member items of a module.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

This command supports the command defaults system.
```

Using Contents on Modules (Module-based)

```
You can run the contents command to:

- List the contents of a module. See 'Understanding Module Hierarchy
  Output' for details.

- List the contents of a particular module version, DesignSync vault
  or legacy configuration so that you can use it for later analysis
  (compare).

  Notes:

 * If a hierarchical reference to a submodule version has been
   overridden by a higher-level href, the hierarchical reference within
   the parent module is NOT considered modified by the contents
   command.
```

* You can also use the contents-foreach function to perform operations on the contents of the output. See the 'contents-foreach' command for more information.

* If filters, views, or hreffilters are specified on the command line, they override all persistent filters, views, and hreffilters applied to the workspace.

Understanding Module Hierarchy Output (Module-based)

To list all modules within a hierarchy, run the contents command on a module with the -recursive option.

- The contents of the top-level (starting) module are listed first, and then the contents of the sub-modules are listed one by one. The order in which the module are listed is not predetermined as the same module can be referenced from multiple points in the module hierarchy.

  - The module version of each sub-module in the hierarchy and its relative path from the top level module is listed. For example, if module Chip references module ALU with a relative path of "alu" and ALU references RAM with a relative path of "submods/ram", then the relative path reported from CHIP to RAM is "alu/submods/ram".

- In a list format output, the submodules are shown in the "modules" property of the top-level module. Each sub-module has a set of properties that includes the module URL, the module version and the relative path.

To list the contents of a module or referenced subdirectories, use the -modulecontext option. If you specify a specific directory within a module, DesignSync returns only the contents of the sub-modules within the specified directory.

For list format output only, each sub-module has a 'modpath' property indicating the path within that module. The 'modpath' property is not shown when the contents of the entire sub-module is listed. For example, if Chip references ALU as above, and you specify: "contents -modulecontext Chip alu/commondir", the contents operation only lists the portion of the ALU sub-module and a "modpath" for all sub-modules in that directory.

The specific module versions reported by the contents command within a module hierarchy depend on the value of hrefmode mode specified. If static mode is specified, the contents reported for the hierarchy follow the static version of the object at the time the href was created. If dynamic mode is specified, the contents resolve the hierarchical references dynamically. If normal mode is specified, the hrefs are followed dynamically until a static selector is reached after which all submodules are resolved statically. The contents command respects the traversal method identified for normal mode by

the "HrefModeChangeWithTopStaticSelector" registry key. For more information, see the "Module Hierarchy" topic in the ENOVIA Synchronicity DesignSync Data Manager User's Guide.

Important: If you are comparing two server-side module URLs, DesignSync uses the value for "HrefModeChangeWithTopStaticSelector" set at the server level, not the user level.

Understanding the path option (Module-based)

When you use the -path option to list the contents of a module, including sub-modules, the relative path reported for members of sub-modules is always relative to the base directory of the module the object is a member of rather than being relative to the base directory of the top level module.

The full relative path to a member from the top module, the path to the member needs to be prepended with the relative path to that module.

When the -path option is used with the -modulecontext option, the relative path reported for objects within sub-modules is either the path from the specified folder or, if the module base directory is below this starting directory, the relative path from the base of the module or sub-module.

When the -fullpath option is used, the paths reported for objects are server addresses in the form <module URL>/<path to object>, for example, "sync://sting:30002/Modules/mymod/subdir/file", where the module URL is the address of the module/sub-module.

Note: This is not a valid individual member address to use in other commands.

The command uses one of two paths:

- A full server URL with an optional selector.
- An optional path (may be a work area or vault folder path) and an optional selector.

  Note: If you specify a module folder as an argument you must use the -modulecontext option.

Using Contents on Legacy Modules (Legacy-based)

You can run the contents command to:

- List folder contents for DesignSync vaults, for example, list all files you created on a new branch.
- List the contents of a legacy module on the server, rather than the

```
one in your work area. This is useful when the module
configuration on the server has changed since you last
fetched it to your work area.

Note: You can also use the contents-foreach function to perform
      operations on the contents of the output. See the
      'contents-foreach' command for more information.
```

Notes for legacy modules (Legacy-based)

```
If no path is given, the path defaults to the module configuration in the
current folder. If no configuration is present, the path defaults to the
'url vault' of current folder and 'url selector' of the path, or the
default configuration if a server path is given.

If a work area path is specified with no configuration, or the
configuration name matches the module configuration in the work area,
then the contents command lists the contents of the configuration the
work area. Otherwise, the command lists the contents of the module
configuration on the server
```

Using Contents on File-Based Objects (File-based)

```
You can run the contents command to list folder contents for
DesignSync vaults, for example, list all files you created on a new
branch.

Note: You can also use the contents-foreach function to perform
operations on the contents of the output. See the 'contents-foreach'
command for more information.
```

SYNOPSIS

```
contents [-exclude <object> [,<object>...]]  [-filter <string>]
         [-format <text| list>] [-fullpath] [-hreffilter <string>]
         [-hrefmode <dynamic|static|normal>]
         [-modulecontext <context>] [-output <file>] [stream <stream>]
         [-[no]path][-[no]recursive]
         [-report {silent | brief | normal | verbose}]
         [-selector <selector>] [-[no]versions]
         [-view view1[,view2,...]] [--] <argument>
```

ARGUMENTS

- Module (Module-based)
- Module Folder (Module-based)
- Workspace or Server Folder (File-based)

```
Specifies one  of the following argument:
```

Module (Module-based)

```
<module>            Specifies the module version or a workspace module
                    for which you want to list the contents.

                    If the version is not specified, use the -selector
                    option. Otherwise, the current selector is used
                    for a workspace module and the default Trunk:Latest
                    is used for a server module.

                    Note: The module contents are always fetched from
                    the server module version.  If a workspace module
                    is specified, the command uses the hierarchy in
                    the module workspace with the module contents on
                    the server.
```

Module Folder (Module-based)

```
<module folder>     Specifies the folder in a module for which
                    you want to list the contents.

                    If the -modulecontext option is specified, listing
                    the contents of a module folder is the same as
                    filtering the contents results to only include that
                    folder.
                    If the -recursive option is specified for a module
                    folder, the sub-modules within the module folder are
                    also specified.
```

Workspace or Server Folder (File-based)

```
<DesignSync object> Specifies the workspace folder or server
                    folder for which you want to list the
                    contents.
```

OPTIONS

- -exclude
- -filter (Module-based)
- -format
- -fullpath
- -hreffilter (Module-based)
- -hrefmode (Module-based)
- -modulecontext (Module-based)
- -output

-exclude

```
-exclude<string>    Specifies the items to exclude from the contents
                    report.

                    If you specify -filter (modules only)and
                    -exclude, then the exclude conditions are applied
                    after filtering the contents thus taking
                    precedence.

                    Note: The global exclude list is added in if
                          either a -exclude or a -filter option is
                          specified, even if the value of the option
                          is an empty string "".
```

-filter (Module-based)

```
-filter <string>    This option filters the objects that are listed as
                    contents. Use this option to specify one or more
                    extended glob-style expressions to identify an exact
                    subset of module objects on which to operate. Use the
                    -exclude option to filter out DesignSync objects that
                    are not module objects.
                    For more information of the -filter option, see the
                    description of filters in the 'ci' command.

                    If you specify the -filter option with a workspace
                    module version, it overrides any persistent
                    filters, views, and hreffilters set on the workspace.
```

-format

```
-format <list       Specifies the format of the output. The format can
    | text>         be:
                    o text -plain text (default)
                    o list -Tcl list structure
```

```
The format when "-format list" is used is a Tcl list
that is designed for further processing, and for easy
conversion to a Tcl array structure. This means that
it is a list structure in name-value pair format. The
top level structure is:
{
   path <path>
   name <path>
   config <config>
   objects <object_list>
   type folder
}

"path" is the starting point path given, and may be
local workspace paths, or a vault path.

"name" is the vault path for which the contents were
fetched. It will match the "path" value if a server
path was specified.

"config" is the configuration being listed, that is
selector value.

object_list is then defined as a list of items of the
form:
{
   name <object name>
   type folder | file
   objects <object_list>
   props <prop_list>
}

The "name" will include the full or relative path if
the -fullpath or -path arguments are used.
For information on how the -path an -fullpath options
are handled when you specify a module as an argument,
see the section 'Understanding the path option'.

The "type" indicates whether the object is a folder or
file (note that collection objects have a type file).
Only a folder has an objects list - this property may
not always be present.

All file objects may have a props list, which is a list
of properties for that object, and has the form:

{
   version <version>
}

A file object has the version property only if the
-versions argument was given.  So, if -versions is not
given, then file objects will not have a props property
-- this property may not always be present. (Although it
may seem unnecessary overhead to have the props sub-list
for a single property, i.e. version, this structure is
maintained for compatibility and for future extensions.)
```

Note that there is no order implied on the objects in the object_lists. In particular, these items may not be sorted. This is different to the text output, where the items within a directory will always be reported in alphanumeric order. The reason for this is that generally the list output will be further processed, and that further processing can decide whether it needs to sort the results.

For legacy modules, any sub-module configuration references are placed immediately below the referencing module.

To process the results, use the contents-foreach function.

-fullpath

-fullpath	Reports the path for each object within the directory as a full URL.

-hreffilter (Module-based)

-hreffilter <string>	Excludes href values during recursive operations on module hierarchies. Because hrefs link to submodules, use -hreffilter to exclude particular submodules when listing module contents. If you specify the -hreffilter option with a workspace module version, it overrides any persistent hreffilters, filters, and views set on the workspace.

-hrefmode (Module-based)

-hrefmode	For a recursive listing of the contents, indicates how the hierarchical reference selectors are followed. Valid values are: o dynamic - Expands hrefs at the time of the listing the contents to identify the submodules to be listed. o static - Lists the contents with the submodules referenced by the hrefs when they were initially created. o normal - Expands the hrefs at the time of the compare operation until it reaches a static selector. If the reference uses a static version, the hrefmode is set to 'static' for the

next level of submodules to be examined; otherwise, the hrefmode remains 'normal' for the next level. (Default). This behavior can be changed using the "HrefModeChangeWithTopStaticSelector" registry key to determine how hrefs are followed.

If a workspace module is specified as the argument with no selector value, then the hierarchy is taken from the workspace and the -hrefmode option is ignored.

-modulecontext (Module-based)

-modulecontext <context>	Specifies the module context, thereby allowing a workspace folder that is below multiple modules to be specified, or allowing a sub-folder of a module on a server to be specified. Note: The module context is required for module folders.

-output

-output <file>	Output the result to the specified file, which is overwritten if it already exists. Default is to send output to the screen if the -format value is text or to return it as the result of the function if the -format value is list.

-path

-[no]path	Controls the path that is reported for each object. Objects are reported on a per-directory basis, with each directory path given as a full URL. The items within the directory can be reported as: - Leaf names (the default, unless '-report silent' is specified) - Relative path to the start of the command (-path). This is the default if '-report silent' is specified. Note: The exclude list affects what objects are reported at the full, relative, or leaf of the path as appropriate. See the -exclude option for details.

-recursive (Module-based)

-[no]recursive Specifies whether to perform this operation i just the specified folder (default) or in its subfolders.

If you use the -recursive option and specify a folder, the contents command operates in a folder-centric fashion. The contents in the folder (all module and non-module data except the hierarchical references) are listed.
The contents listed can be further refined using the -filter, -hreffilter or -exclude options.

If you use the -recursive option and specify a module, the contents command operates in a module-centric fashion. The contents in the folder and all subfolders (that is all module and non-module data along with the hierarchical references) are listed.
The contents listed can be further refined using the -filter, -hreffilter or -exclude options.

Notes:
- o Hierarchical references are followed only for modules. All hrefs to legacy modules, DesignSync vaults or IPGear deliverables are skipped.
- o The -nomodulerecursive option has been deprecated. To filter modules, use the -hreffilter option.

-recursive (Legacy-based)

-[no]recursive Specifies whether to perform this operation i just the specified folder (default) or in its subfolders.

If you use the -recursive option and specify a folder, the contents command operates in a folder-centric fashion. The contents in the folder (all module and non-module data except the hierarchical references) are listed.
The contents listed can be further refined using the -exclude option.

If you use the -recursive option and specify a module, the contents command operates in a module-centric fashion. The contents in the folder and all subfolders (that is all module and non-module data along with the hierarchical references) are listed.

The contents listed can be further refined using the -exclude option.

Notes:
- Hierarchical references are followed only for modules. All hrefs to legacy modules, DesignSync vaults or IPGear deliverables are skipped.
- The -nomodulerecursive option has been deprecated. To operate in non-recursive mode, Use the -norecursive option.

-recursive (File-based)

-[no]recursive — Specifies whether to perform this operation i just the specified folder (default) or in its subfolders.

If you use the -recursive option and specify a folder, the contents command operates in a folder-centric fashion. The contents listed can be further refined using the -exclude option.

-report

-report <mode> — Controls level of additional information reported as the command progresses.

Valid values are:

- silent - Only the primary output is given - the list of objects and versions. In this case, for the text output, the default is to show the relative path name (-path).
- brief - Displays the same information as 'normal'.
- normal - Include header information and progress lines where a long command might be performed, such as when scanning the vault. (Default)
- verbose - Include information on configuration mappings.

-selector (Module-based)

-selector — A valid selector or selector list.

<selector>	You should distinguish between branch and version selectors. If you are specifying a branch other than Trunk, specify both the branch and version as follows: '<branchtag>:<versiontag>', for example, 'Rel2:Latest'. You can also use the shortcut, '<branchtag>:', for example, 'contents -selector B1:'.

-selector (File-based)

-selector <selector>	A valid selector or selector list. Note: The selector cannot contain Date() or VaultDate() items. You should distinguish between branch and version selectors. If you are specifying a branch other than Trunk, specify both the branch and version as follows: '<branchtag>:<versiontag>', for example, 'Rel2:Latest'. You can also use the shortcut, '<branchtag>:', for example, 'contents -selector B1:'.

-stream

-stream <stream>	Output to the given stream (which should be the result of a Tcl 'open' function call). Default is to send output to the screen if the -format value is text or to return it as the result of the function if the -format value is list.

-version

-[no]versions	Include the version numbers for the objects listed.

-view (Module-based)

-view view1 [,view#[,...]]	Species the view name or list of view names used when retrieving the contents of the module. The view list must be specified as a comma-separated list. Note: When the -view option is used with a

```
                    workspace module instance argument, you must also
                    specify the -selector option to identify a
                    server-side module version on which to operate.

                     If you specify the -view option with a workspace
                     module version, it overrides any persistent
                     views, filters, and hreffilters on the workspace.

                    Specifying the view name "none" specifies that the
                    command should not apply any views to the
                    contents.
```

--

```
 --                 The command option '--' indicates that following
                    arguments should not be taken as options, but as
                    paths that begin with a '-'.
```

RETURN VALUE

```
Empty string if -format value is text.
Tcl list if the -format value is list.
Empty string if -output or -stream is used with -format.

When run from a server-side script, the server-side URL used in the
results is relative to the server root. For example, sync:///@Trunk:Latest.
The {host}:{port} is omitted.  If the server-side script is run from a
browser (via a ProjectSync URL), then the script must format the output
for display within an HTML page.
```

SEE ALSO

```
contents-foreach, ls, ls-foreach, compare, compare-foreach, showstatus,
command defaults
```

EXAMPLES

- Example Showing Contents of Server for Current Working Directory
- Example Showing Contents Output to a Stream
- Example Showing Contents of a Module Instance (Module-based)
- Example Showing Contents of Server Module Version (Module-based)
- Example Showing Contents of a Legacy Module Configuration (Legacy-based)

Example Showing Contents of Server for Current Working Directory

```
This example shows the contents of the server (vault or module)
associated with the current workspace.
```

```
  dss> contents
```

This example shows the same contents, but includes version numbers and paths in the output.

```
   dss> contents -config Rel1 -recursive -fullpath -versions

Gathering configuration data from vault
 sync://svr1.ABCo.com:30002/Projects/P1@Rel1

Contents of:
file:///home/users/username/myprojects/P1
Vault:
sync://svr1.ABCo.com:30002/Projects/P1@Rel1

Version Object Name
1.2      sync://svr1.ABCo.com:30002/Projects/P1/file1.txt
1.4      sync://svr1.ABCo.com:30002/Projects/P1/file2.txt
1.1      sync://svr1.ABCo.com:30002/Projects/P1/file4.txt
1.5      sync://svr1.ABCo.com:30002/Projects/P1/file5.txt

sync://svr1.ABCo.com:30002/Projects/P1/subdir@Rel1

Version Object Name
1.5     sync://svr1.ABCo.com:30002/Projects/P1/subdir/file7.txt
1.2     sync://svr1.ABCo.com:30002/Projects/P1/subdir/file8.txt
```

Note: In this example, if the -fullpath option were not specified, then the relative path would be used for the Object Name.

Example Showing Contents Output to a Stream

This example shows the contents of the configuration Rel4, for the vault directory structure starting at the vault location given. It sends the output results to the specified open stream.

```
  dss> contents -config Rel4 sync://svr1.ABCo.com:30002/Projects/P1
       -stream $p
```

Example Showing Contents of a Module Instance (Module-based)

This example shows the contents of a workspace instance.

```
  dss> contents ModuleA%2
```

Example Showing Contents of Server Module Version (Module-based)

This example shows the contents of a server module version.

```
dss> contents sync://srv2.ABCo.com:2647/Modules/Mod1
 Gathering data from vault
 sync://srv2.ABCo.com:2647/Modules/Mod1@Trunk:Latest

 Module: sync://srv2.ABCo.com:2647/Modules/Mod1@1.2
 Contents of folder: /

 Object Name
 -----------
 File1.txt
 File2.txt
 File3.txt
```

Example Showing Contents of a Legacy Module Configuration (Legacy-based)

```
This example shows the contents of configuration Wa1 for the vault
folder associated with the current specified argument.

dss> contents -recursive -version /home/workareas/WA1
 Gathering configuration data from vault
 sync://srvr1.ABCo.com:2647/Projects/Top@Alpha
 Gathering configuration data from vault
 sync://srvr3.ABCo.com:2647/Projects/ALU@Rel1
 Gathering configuration data from vault
 sync://srvr5.ABCo.com:2647/Projects/Decoder@Rel1

 Contents of:
 file:///home/workareas/WA1
 Vault:
 sync://srvr1.ABCo.com:2647/Projects/Top@Alpha

 Version   Object Name
 -------   -----------
 1.3       top.v

 sync://srvr1.ABCo.com:2647/Projects/Top/projdoc@Alpha

 Version   Object Name
 -------   -----------
 1.3       projinfo.txt
 1.4       projlist.txt

 sync://srvr3.ABCo.com:2647/Projects/Top/submods/ALU@Rel1

 Version   Object Name
 -------   -----------
 1.3       alu.v
 1.4       mult8.v

 sync://srvr5.ABCo.com:2647/Projects/Top/submods/Decoder@Rel1

 Version   Object Name
 -------   -----------
 1.2       decoder.v
```

```
The contents command output shows the configuration as it exists in
the work area, rather than vault. For example, the output shows the
Top@Alpha configuration in the WA1 work area as containing the
ALU@Rel1 configuration, whereas in the vault, Top@Alpha contains
ALU@Rel2.

When listing objects, however, the contents command lists object
versions as they exist in the vault. For example, the command
output lists version 1.3 of top.v because version 1.3 is in the
Top@Alpha configuration in the vault.
```

url contents Command

NAME

```
url contents        - Returns the objects in a container object
```

DESCRIPTION

- Notes for Modules (Module-based)

```
This command returns a list of URLs of the objects contained in the
specified container object, such as a folder or configuration.  If
the object is not appropriate for the requested operation, an empty
list is returned.

The 'url contents' command is not recursive. For example, 'url
contents' on a ProjectSync configuration always returns folders
as part of the configuration. You can then invoke 'url contents'
on each subfolder in the project.

Note: In stcl/stclc mode, when specifying a URL that contains a
semicolon (;), surround the URL with double quotes.
```

Notes for Modules (Module-based)

```
The "url content" command shows the contents of a module folder, but
is not used for examining the contents of a module and therefore does
not accept module as a valid argument type. Use "ls" and "contents"
commands to list the full module contents.
```

SYNOPSIS

```
url contents [-all | -ifpopulated [-incremental]] [-prefetch]
             [-version <selector>[,<selector>...]] [--] <argument>
```

ARGUMENTS


```
Specify one or more of the following arguments:
```

Module Folder (Module-based)

```
<module folder>      Returns a list of the URLs of files and folders
                     contained in the specified workspace module
                     folder.
```

DesignSync Folder (File-based)

```
<DesignSync folder>  Returns a list of URLs of files and folders
                     contained in the specified folder in the
                     workspace.
```

DesignSync Vault (File-based)

```
<DesignSync vault>   Returns a list of the URLs of the different
                     vault versions checked into the specified vault.
```

OPTIONS

-all

```
-all                 Reports the objects in the local folder as well
                     as those objects that would be there if it were
                    fully populated with the contents of the
                    associated vault. If the object is a vault-side
                    object (a vault, version, or branch), this
                    option is ignored.

                     This option is mutually exclusive with
```

-ifpopulate.

-ifpopulated

-ifpopulated	Report the contents of the local folder if it were fully populated with the contents of its associated vault. You can also specify -incremental to limit the result to return only those objects that would return in an incremental populate as opposed to a full populate. The list is empty if the object is not a local folder. This option is mutually exclusive with -all.

-incremental

-incremental	Modifies -ifpopulated to limit the result to return the URLs of only those objects that would return in an -incremental populate. Note: You must have at some time performed a full populate on the folder for the -incremental option to work properly.

-prefetch

-prefetch	Used for advanced programming; exposes an optimization to the caller. If used, the call to contents is slower, but the subsequent enumeration of the returned list has extra information cached. If the caller needs to enumerate the contents and for each object call commands such as 'url tags' or 'url properties', overall performance is better if this option is used. If the caller needs to retrieve only the names of the objects, this option makes the operation slower.

-version

-version <selector>	Use with -ifpopulate or -all. Specifies the selector list (typically branch or version tag) to use for the hypothetical populate. The default (-version not specified) is to inherit the selector from the parent folder.

Note: To use -version to specify a branch, specify both the branch and version as follows: '<branchtag>:<versiontag>', for example, 'Rel2:Latest'. You can also use the shortcut, '<branchtag>:', for example "Rel2:". If you don't explicitly specify the branch selector in this way, DesignSync does not resolve the selector as a branch selector.

--

-- Indicates that the command should stop looking for command options. Use this option when the argument to the command begins with a hyphen (-).

RETURN VALUE

For a client-side folder (Asic): Returns list of client-side folders and files (file://home/karen/Asic/Sub file://home/karen/Asic/x.v).

For a vault ("sync://holzt:2647/Projects/Asic/x.v;"): Returns vault versions: ({sync://holtz:2647/Projects/Asic/x.v;1.1}
{sync://holtz:2647/Projects/Asic/x.v;1.2}
{sync://holtz:2647/Projects/Asic/x.v;1.3}).

For a version ("sync://holzt:2647/Projects/Asic/x.v;1.1", "sync://holzt:2647/Projects/Asic/x.v;yellow"): Not a container object; returns an empty list.

For branch ("sync://holzt:2647/Projects/Asic/x.v;1.5.1", "sync://holzt:2647/Projects/Asic/x.v;Golden:Latest"): Not a container object; returns an empty list.

For a project (sync://holzt:2647/Projects/Asic): Returns vault folder containing project: (sync://holzt:2647/Projects).

For a configuration (sync://holzt:2647/Projects/Asic/Sub@Rel1): Returns vault folder: (sync://holzt:2647/Projects/Asic).

For a server-side note system URL (sync:///Note)
Returns the list of note systems on the server (currently the only note system is SyncNotes)

For a server-side SyncNotes URL (sync:///Note/SyncNotes):
Returns the list of URLs for all note types defined on the server.

For a server-side note-type URL (sync:///Note/SyncNotes/HW-Defect-1):
Returns a list of URLs of all notes of type HW-Defect-1.

For a server-side Users URL (sync:///Users)
Returns a list of URLs for all user profiles on the server.

```
For a module folder, returns a list of members in that folder.

For other objects: Returns an empty list.
```

SEE ALSO

```
note systems, notetype enumerate, populate, selectors, url container,
url notes, url users, url versions
```

EXAMPLES

- Example Showing the Contents of a Module Folder (Module-based)
- Sample File Structure for Examples (File-based)
- Example of Local Folder Contents (File-based)
- Example of Vault Folder Contents (File-based)
- Example of Returning the Contents of a Branch (File-based)
- Example Showing the Contents Resulting from Full Populate (File-based)
- Example Showing the Contents Resulting From Incremental Populate (File-based)
- Example Showing Contents Resulting From Populate with Selector (File-based)
- Example Showing Contents Resulting from Populate with Configuration (File-based)

Example Showing the Contents of a Module Folder (Module-based)

```
In the following example, the workspace //MyModules contains the
following:
  MyModules
    File1.txt
    File2.txt
    File3.txt
    File4.txt
Return contents of local folder MyModules

  stcl> url contents /home/tachatterjee/MyModules
file:///home/tachatterjee/MyModules/File1.txt
file:///home/tachatterjee/MyModules/File2.txt
file:///home/tachatterjee/MyModules/File3.txt
file:///home/tachatterjee/MyModules/File4.txt
```

Sample File Structure for Examples (File-based)

```
In the following examples, the local work area
/Projects/ASIC contains the following:
  ASIC
    FileA          # Three versions exist in the vault
    FileB          # There is a new version in the vault
    FileC          # Not under revision control
```

```
    FileD          # In the vault, but not in local working directory
    Decoder
      FileE        # There is a new version in the vault
      FileF        # Only file to have tag 'Gold' (version 1.2)
```

Example of Local Folder Contents (File-based)

```
Return contents of local folder ASIC

  dss> url contents /Projects/ASIC
  file:///Projects/ASIC/Decoder
  file:///Projects/ASIC/FileA
  file:///Projects/ASIC/FileB
  file:///Projects/ASIC/FileC
```

Example of Vault Folder Contents (File-based)

```
Return contents of vault folder ASIC

  dss> url contents sync://holzt:2647/Projects/ASIC
  sync://holzt:2647/Projects/ASIC/Decoder
  sync://holzt:2647/Projects/ASIC/FileA;
  sync://holzt:2647/Projects/ASIC/FileB;
  sync://holzt:2647/Projects/ASIC/FileD;
```

Example of Returning the Contents of a Branch (File-based)

```
Return contents of FileA main branch

  dss> url contents "sync://holzt:2647/Projects/ASIC/FileA;1"
  sync://holzt:2647/Projects/ASIC/FileA;1.1
  sync://holzt:2647/Projects/ASIC/FileA;1.2
  sync://holzt:2647/Projects/ASIC/FileA;1.3
```

Example Showing the Contents Resulting from Full Populate (File-based)

```
Return contents of /Project/ASIC resulting from full populate

  dss> url contents -ifpopulated /Projects/ASIC
  file:///Projects/ASIC/Decoder
  file:///Projects/ASIC/FileA
  file:///Projects/ASIC/FileB
  file:///Projects/ASIC/FileD
```

Example Showing the Contents Resulting From Incremental Populate (File-based)

```
 Return updated objects of /Project/ASIC after incremental populate

  dss> url contents -ifpopulated -incremental /Projects/ASIC
  file:///Projects/ASIC/Decoder
  file:///Projects/ASIC/FileB
  file:///Projects/ASIC/FileD
```

Example Showing Contents Resulting From Populate with Selector (File-based)

```
Return what is populated when the selector is "Gold", which can be a
version or branch tag

  dss> url contents -ifpopulated -version Gold /Projects/ASIC
  file:///Projects/ASIC/Decoder
```

Example Showing Contents Resulting from Populate with Configuration (File-based)

```
Return the contents of the ASIC project's "Gold" configuration

  dss> url contents sync://holzt:2647/Projects/ASIC@Gold
  sync://holzt:2647/Projects/ASIC/Decoder@Gold

  dss> url contents sync://holzt:2647/Projects/ASIC/Decoder@Gold
  sync://holzt:2647/Projects/ASIC/Decoder/FileF;1.2
```

contents-foreach

contents-foreach Command

NAME

```
contents-foreach    - Function to process the results of a contents
                      command
```

DESCRIPTION

```
This routine loops over the items in a "contents" results list, and
processes each item in turn.

Note: The only property types typically available for each object
are, name, type, and version. The name and type properties are always
present in the contents output; "versions" is only present when the
"-versions" option was specified to the contents command.
```

SYNOPSIS

```
contents-foreach var result_list tcl_script [-nofolder] [-path]
```

ARGUMENTS

- var
- results_list
- tcl_script

var

var	This is the loop variable. It is treated as a Tcl array, and on each loop around contains the set of properties for the next object in the result_list. In addition to the properties in the "props" value for each object (i.e. the version), the array will contain a "name" property and a "type" property, which are the name and type properties for the object. Note: For modules, the contents-foreach function returns a value of "Module" for the "type" property.

results_list

result_list	This is the list of objects to be processed. It must be the result value of a call to the "contents" command with the "-format list" option.

tcl_script

tcl_script	This is the piece of Tcl code that is executed on each loop.

OPTIONS

- -nofolder
- -path

-nofolder

-nofolder	If specified, then the tcl_script will not be called for Folder type objects in the result_list.

-path

```
-path           The "name" property on each loop is usually just the
                "name" property for the object. However, if this
                option is specified, and a recursive "contents" was
                performed, then the "name" property is the relative
                path to each object. Normally, you would run
                "contents" with the -path or -fullpath option, in
                which case the "name" property contains an
                appropriate relative or full path. If you did not
                do that, then passing the "-path" option to
                contents-foreach will mean that the "name" property
                contains the relative path for each item, thus
                allowing you to differentiate between items with the
                same name in different folders.
```

SEE ALSO

```
contents
```

EXAMPLE

```
This example shows using processing the compare-foreach command to
process the results from a compare command.

 set result_list [contents -selector RelA:Latest -version -rec -format list]

 contents-foreach obj $result_list -nofolder { puts "Object: $obj(name),
 version: $obj(version)" }
```

datasheet

datasheet Command

NAME

```
datasheet           - Displays an object's data sheet
```

DESCRIPTION

```
This command displays the data sheet for the specified object.
The information that is displayed depends on the object type. For
example, the data sheet for a file in your working folder contains
information such as lock status, modification status, version
number, and associated tags.
```

DesignSync displays the information in a browser window. On Windows platforms your system's default browser is used, and the data sheet is displayed in an existing browser window if one is available. On UNIX, the browser is determined by a registry setting in one of the DesignSync client registry files. You can override the installation- or site-wide default browser using the SyncAdmin tool. On UNIX, DesignSync invokes a new browser to display the data sheet even if you have a browser already running.

This command is subject to access controls on the server. See the ENOVIA Synchronicity Access Control Guide for details.

SYNOPSIS

```
datasheet <object>
```

OPTIONS

None.

RETURN VALUE

None.

EXAMPLES

This example displays the data sheet for top.v.

```
dss> scd Projects/Sportster/top
dss> datasheet top.v
```

This example displays the data sheet for the top.v vault.

```
stcl> datasheet "sync://holzt:2647/Projects/Sportster/top/top.v;"
```

This example displays the data sheet for version 1.2 of top.v.

```
stcl> datasheet [url vault top.v]1.2
```

diff

diff Command

NAME

```
diff                - Compares files and versions of files
```

DESCRIPTION


```
The file/version comparison facility has three components:
  - The DesignSync 'diff' command, which is documented here.
  - The DesignSync GUI Tools->Compare Files commands. Performing
    comparisons from the GUI simplifies the selection of files and
    versions for comparison, and also provides shortcuts to
    perform the most common types of comparisons. The GUI commands
    ultimately invoke the DesignSync diff command. See DesignSync
    Help for details on the Compare Files commands.
  - The graphical interface. DesignSync provides the capability to
    display diff results in a graphical diff client.
    Note: The graphical interface client does not directly understand
    DesignSync versions; you must invoke it from the DesignSync
    'diff' command using the -gui option, or the DesignSync GUI
    Advanced Comparison dialog in order to compare versions.
```

Notes for Collection Objects

```
The built-in diff command supports comparing cell views members from
different versions of a cell view, or comparing local modification of
a cell view to different server versions of the cell view.

Important: When specifying collection objects as the files to
compare, you must use the -member option to specify the relative path
of the cell view.  This provides the location on the server of the
cell view within the collection and indicates to DesignSync that the
object is a cell view within a collection. To find the relative path
of a cell view, use the url members command with the -relative option.
```

Note for Modules (Module-based)

```
Note:  The diff command is for comparing files.  To compare module
contents, use the compare command.
```

SYNOPSIS

```
diff [-ancestor <commonAncestorFile>] [-binary] [-case] [-embed]
     [-kk] [-member <cellview_path>] [-modulecontext <context>]
     [-standard | -unified | -syncdiff | -annotate | -gui]
     [-output <resultFile>] [-[no]usemoduleversions] [-version <id>]
     [-white] {[-file1] <fileA>} [[-file2] <fileB>]  [--]
```

ARGUMENTS

File Object (Module-based)

<file>	You can specify one or two files as simple filenames, relative pathnames, absolute pathnames, URLs to module members, DesignSync objects, collection cell view versions, and legacy module members. You can also specify versions by appending the filename with a semicolon (;) and a version number or tag name (including Latest and Orig). Note: When in stcl/stclc mode, you must surround filenames that have spaces or version-extended names with double quotes, for example: "foo.bar;1.5"The file argument can be specified with or without the -file/-file2 options. You can use version-extended filenames (see Description section).

File Object (File-based)

<file>	You can specify one or two files as simple filenames, relative pathnames, absolute pathnames, collection cell view versions, or URLs to DesignSync objects. You can also specify versions by appending the filename with a semicolon (;) and a version number or tag name (including Latest and Orig). Note: When in stcl/stclc mode, you must surround filenames that have spaces or version-extended names with double quotes, for example: "foo.bar;1.5"The file argument can be specified with or without the -file/-file2 options. You can use version-extended filenames (see Description section).

OPTIONS

-ancestor (Module-based)

```
-ancestor <fn>   Specifies the common ancestor for a three-way
                 comparison. You can use version-extended filenames
                 (see Description section).

                 For example, two users fetch the same version of a
                 file, and each makes changes to their copy. By
                 specifying the original unmodified version as the
                 common ancestor, the first user's modified copy as
                 fileA and the second user's modified copy as fileB,
                 the diff operation can indicate who made which
                 changes and whether the changes would conflict when
                 merged.

                 Specify an asterisk as the ancestor (-ancestor *) to
                 have diff automatically calculate the closest common
                 ancestor of the two file versions (using 'url
                 resolveancestor').  This option is always a file
                 version.

                 When -usemoduleversions is specified with the
                 -ancestor * option, the module version is used to
                 identify file versions for the closest common
                 ancestor calculation. The -ancestor option does not
                 identify the closest common ancestor of the module
                 versions themselves.

                 Note: Only -syncdiff and -annotate output formats
                 support three-way comparisons.
```

-ancestor (File-based)

-ancestor <fn> Specifies the common ancestor for a three-way comparison. You can use version-extended filenames (see Description section).

For example, two users fetch the same version of a file, and each makes changes to their copy. By specifying the original unmodified version as the common ancestor, the first user's modified copy as fileA and the second user's modified copy as fileB, the diff operation can indicate who made which changes and whether the changes would conflict when merged.

Specify an asterisk as the ancestor (-ancestor *) to have diff automatically calculate the closest common ancestor of the two file versions (using 'url resolveancestor'). This option is always a file version.

Note: Only -syncdiff and -annotate output formats support three-way comparisons.

-annotate

-annotate Uses a column format to display the entire comparison text. The first column displays the line number in fileA. The second column displays the line number in fileB. The third column indicates whether the line is changed and what has changed. The final column provides the text of the line indicated.

This option is mutually exclusive with the -gui, -standard, -syncdiff, and -unified options.

-binary

-binary Performs a fast comparison of the files, reporting only whether the files are identical or not. Because differences between binary files cannot be reported, a fast binary comparison is automatically performed if 'diff' detects that either of the files being compared is a binary file. Note that the -kk, -embed, -white, and -case diff options are ignored when performing a binary comparison.

-case

```
 -case            Ignore character case differences.
```

-embed

```
 -embed           Ignore differences in the amount of whitespace
                  within a line. For example, using -embed, there
                  is no difference between a sentence with one
                  space between each word and the same sentence with
                  three spaces between each word.
```

-file1

```
 -file1 <fileA>   Specifies the first file or version to be
                  compared. In most cases, this should be the older
                  version of the two being compared. For more
                 information about what can be specified for <fileA>,
                 see the <file> argument. If you do not specify the
                 fileB argument, then fileA is compared to its
                 Original (;Orig) version, which is the version that
                 you checked out prior to making local
                 modifications.

                  Note: The -file1 and -file2 switches are optional;
                  you can specify fileA and fileB without the
                  switches. However, you must either specify both
                  switches or omit both switches. The following
                  syntax is invalid:
                     diff -file1 a.txt b.txt
```

-file2

```
 -file2 <fileB>   Specifies the second file or version to be
                  compared.  For more information about what can be
                 specified for <fileA>, see the <file> argument. If
                 omitted, fileB is assumed to be the Original (;Orig)
                 version of fileA.

                  FileB should generally be the version with the more
                  recent changes. If fileB is older than fileA, then
                  the comparison succeeds, but the results are the
                  inverse of the actual modifications. For example,
                  if you add a line to the newer file, but specify
                  the newer file as fileA, then diff reports that
                  this line was deleted from fileB rather than
                  indicating that the line was added to fileA. In
                  some cases, this inverse report is useful; for
                  example, when backing out a set of changes.
```

-gui

-gui

Invokes the defined graphical Diff utility to display the comparison results. DesignSync provides a graphical utility, or, if you have a preferred diff tool, you may configure your system to use that tool.

For information configuring DesignSync to recognize your graphical Diff utility, see the ENOVIA DesignSync Administrator's Guide.

Note: The -kk, -embed, -white, and -case diff options are controlled by registry keys for the graphical Diff utilities. The registry key settings override any command line options specified. The -output option is ignored.

This option is mutually exclusive with the -annotate, -standard, -syncdiff, and -unified options.

-kk

-kk

Stands for "keep keywords", ignores differences in RCE keyword values by hiding the keyword values (collapsing the keywords) prior to comparing the files. RCE keywords are tokens, such as $Revision$, $Author$, and Log (see Notes), that you can add to your files to provide revision information, such as revision number, author, and comment log.

For example, if the first lines of fileA and fileB are:

```
   fileA:   $Revision: 1.1 $
   fileB:   $Revision: 1.3 $
```

then diff reports the difference unless you specify -kk, in which case diff collapses each line to:

```
   $Revision$.
```

Differences in keyword usage and placement are always reported. For example:

```
   fileA:   $Revision: 1.1 $
   fileB:   $Author: Goss $
```

diff reports the difference irrespective of whether you specify -kk because the keywords themselves, not just the keyword values, are different.

Notes:

- Log, when expanded, permanently adds log information to your files. The -kk option does not hide these log messages prior to performing a comparison. Diff programs such as tkdiff may flag differences or conflicts

```
                    (if log information has been edited by hand)
                    if you use $Log$ in your files.
                  - The diff command honors the $KeysEnd$ keyword;
                    any expanded keywords after $KeysEnd$ are compared
                    fully and literally.
```

-member

```
 -member            Specifies the relative path of the collection object
 <collection_path> member. This option is used to indicate to the
                    system that the files specified are within a
                   collection and to provide the relative path to the
                   cell view.

                    Note: If you have specified a cell view within a
                    collection as the file argument, this option is
                    required.
```

-modulecontext (Module-based)

```
-modulecontext     Specifies a module context to be used for file
 <context>         arguments that are not present in the workspace.
                   This allows you compare files that are not present
                   in your workspace.
                   Note:  If the file is not present in the workspace,
                   you must specify the full natural path of the module
                   member.

                   You can only specify one module context, so if you
                   are using the -modulecontext option and specifying
                   files in two different modules, at least one of the
                   files must be present in your workspace.
```

-output

```
 -output <fn>      Specifies an output file for the diff results. By
                   default, the results are displayed in the
                   shell window. The -output option is ignored if you
                   specify -gui.

                   Caution: Any existing file of the same
                   name is overwritten without warning.
```

-standard

```
-standard          Displays differences in standard Unix diff format.

                   This option is mutually exclusive with the
```

```
                -annotate, -gui, -syncdiff, and -unified options.
```

-syncdiff

```
 -syncdiff      Displays the changed text with margin
                annotations indicating the changes.

                This option is mutually exclusive with the
                -annotate, -guide, -standard, and -unified options.
```

-unified

```
 -unified       Displays differences in unified diff format.

                This option is mutually exclusive with the
                -annotate, -gui, -standard, and -syncdiff options.
```

-usemoduleversions (Module-based)

```
 -[no]usemoduleversions
                Indicates whether the specified version applies
                to the file being compared (-nousemoduleversions) or
                the module being compared (-usemoduleversions.)

                -nousemoduleversions uses the specified version to
                refer to the file version. (Default) That file may
                not be a member of a module, or may be a member of a
                module version with a different version number.  For
                example: FileA;1.4 might be a member of module
                Chip;1.20)

                -usemoduleversions uses the specified version to
                refer to module version which contains the file. For
                example, specifying FileA;1.20 with the
                usemoduleversions option identifies that module
                version 1.20 contains version 1.4 of FileA, and the
                diff runs against file version 1.4.

                Notes:
                o When -usemoduleversions is used, the output of the
                  command always provides the version information
                  for the specified file.

                   o When -moduleversion is used with the
                     -ancestor * option, which specifies the common
                     ancestor of two file versions, the
                     moduleversion is resolved to the file version
                     before attempting to resolve the ancestor.  The
                     common ancestor is returned as a file version
                     of the individual file, not as a module
```

version.

-version (Module-based)

-version <selector>	Specifies another version of fileA to compare to to fileA. If the selector resolves to a branch, the Latest version on that branch is used for the comparison. When using the -version option, you only need to specify fileA for comparison. The system implicitly processes the two specified versions of fileA without requiring you to type one version as fileB. Note: When the -version option is used with the -usemoduleversions option, the file is compared against the file contained in the module version specified.

-version (File-based)

-version <selector>	Specifies another version of fileA to compare to to fileA. If the selector resolves to a branch, the Latest version on that branch is used for the comparison. When using the -version option, you only need to specify fileA for comparison. The system implicitly processes the two specified versions of fileA without requiring you to type one version as fileB.

-white

-white	Ignore leading and trailing whitespace. For example, using -white, there is no difference between UNIX and PC line endings, or different indentation levels, as long as the rest of the line content matches.

--

--	Indicates that the command should stop looking for command options. Use this option when arguments to the command begin with a hyphen (-).

SEE ALSO

```
DesSync, url resolveancestor, keywords
```

EXAMPLES

Examples of Comparing a File against the Original Version

```
Example showing compare of a working copy of a file against the
original version to see what changes you have made. All of the
following specifications are equivalent:

  dss> diff foo.bar
  dss> diff "foo.bar;Orig" foo.bar
  dss> diff -v Orig foo.bar
```

Examples of Comparing a File Against the Latest Server Version

```
Example showing compare of a working copy against the latest version
on the same branch, using unified diff format:

  dss> diff -unified foo.bar "foo.bar;Latest"
  dss> diff -unified -version Latest foo.bar
```

Example of Comparing a File Against A Specified Version

```
Example showing compare of a working copy of a file on
the Trunk branch against the latest version on the "rel30" branch:

  dss> ls -report H samp.asm
    Branch Tags  Name
    -----------  ----
          Trunk  samp.asm
  dss> diff -v rel30: samp.asm
```

Example of Comparing Original File Against Latest Server Version

```
This example shows how the latest server version of the file differs
from the original version.

  dss> diff "foo.bar;Orig" "foo.bar;Latest"
```

Example of Showing Conflicts in Your Local Version

```
This example shows how to find conflicts between your locally
modified copy and the latest checked-in version:

  dss> diff -ancestor "foo.bar;Orig" foo.bar "foo.bar;Latest"

or equivalently, use the "*" notation to have "diff" calculate
the common ancestor automatically:

  dss> diff -ancestor * foo.bar "foo.bar;Latest"
```

Examples of Comparing Collection Cell View Versions

```
These examples show different ways of specifying the same comparison
of member file from different versions of a cell view version.

stcl> diff -member verilog/verilog.v [url vault verilog.sync.cds]1.2 \
[url vault verilog.sync.cds]1.3

stcl> diff -member verilog/verilog.v -version 1.2 \
{verilog.sync.cds;1.3}

stcl> diff -member verilog/verilog.v {verilog.sync.cds;1.2} \
{verilog.sync.cds;1.3}
```

Example of Comparing Against the Local Cell View Version

```
This example shows comparing the local version of a member with a
specified cell view version.

stcl> diff -member verilog/verilog.v {verilog.sync.cds;1.3} \
verilog.sync.cds
```

Example of Comparing Files Using the Module Version (Module-based)

```
This example specifies using the module version 1.18.  The Chip module
```

```
version 1.18 contains version 1.1 of the test.c file.

  dss> diff -usemoduleversions -version 1.18 test.c
 NOTE: Object test.c, module version 1.18, mapped to object version 1.1
 7c7
 <           printf ("Hello Big World!\n");
 ---
 >           printf ("Hello World!\n");
 1 Differences detected
```

Example of Comparing Files Using the Member Version (Module-based)

```
This version of the diff command uses the same command as the Example
of Comparing Files Using the Module Version example, but doesn't
specify -usemoduleversion (You could also specify
-nousemoduleversion) so it uses version 1.18 of temp.c regardless of
the module version.

dss> diff -version 1.18 test.c
 som-E-152: No Such Version.

Because there is no version 1.18 of the test.c file, this example,
unlike the previous example, generates an error.  Using the correct
module member version, 1.1; however, results in comparing the 1.1
version of test.c on the server to the modified local version in the
workspace, just as was done in the previous example.

dss> diff -version 1.1 test.c
 7c7
 <           printf ("Hello Big World!\n");
 ---
 >           printf ("Hello World!\n");
 1 Differences detected
```

Example Comparing a Module Member to a Non-Local Module Member (Module-based)

```
This example shows comparing a module member in the workspace, with a
module member not in the workspace:

   dss> diff -modulecontext Chip foo.bar "/oldfoo.bar;1.4"

 Note: FileA, foo.bar, is not a full natural path so the system uses
 the workspace object.  FileB is specified with a full natural path,
 and is found using the module context.
```

Example of Specifying the Module Version with the Ancestor * Option (Module-based)

```
This example shows the difference in resolving -ancestor * when
-usemoduleversion is used to specify the module version.
```

```
dss> diff -ancestor * -modulecontext  \
     sync://srv2.ABCo.com:2647/Modules/Chip -usemoduleversion  \
     -version 1.20 test.c
 NOTE: Object test.c, module version 1.20, mapped to object version 1.3
 Note: Three-way diffs cannot be displayed in standard mode. Using
 syncdiff mode.
 Comparing: (A => B, C)
     (Ancestor) sync://srv2.ABCo.com:2647/Modules/Chip/vault/f7/ \
                f75e11f54656d4c28bb9f37fef1b55f5;1.2
     (B)        file:///home/rsmith/MyModules/chipDiff/test.c
     (C)        sync://srv2.ABCo.com:2647/Modules/Chip/vault/f7/ \
               f75e11f54656d4c28bb9f37fef1b55f5;1.3
 Deleted from B & C (A6, B6, C6)            printf ("Hello Big World!\n");
 Unresolved conflict (A6, B6, C6)            printf ("Hello Big Giant
 World!\n");
 -------
             printf ("Hello Big Tiny World!\n");
 2 Differences detected
```

help

help Command

NAME

```
help                - Provides help on the Synchronicity command set
```

DESCRIPTION

```
This command provides a variety of help related functions,
displaying the information in the output window. Help is available
for:
 - All DesignSync command-line commands
 - DesignSync topics such as using wildcards or running server-side
   scripts
 - ProjectSync command-line commands

For compound commands such as the 'url' and 'note'
commands, surround the command with double quotes and put
exactly one space between the two keywords of the command (see
Example section).

Every DesignSync command has '-help', '-?', and '-usage' options
that you can specify to get full or brief help.

Note:
   You can access other DesignSync and related products and
   integrations documentation from the DesignSync Documentation Main
   Menu:

   - (Windows only) Select "DesignSync Documentation" from the
```

```
      Windows Start menu, typically:

        Start->Programs->Dassault Systems DesignSync <version>->
          DesignSync Documentation

    - Enter the following URL from your Web browser:

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

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

    - Enter the following URL from your Web browser:

        file:///$SYNC_DIR/share/content/doc/index.html

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

SYNOPSIS

```
help  [-all] [-brief] [-output <file>] [-summary] [<topic> [...]]
```

OPTIONS

- -all
- -brief
- -output
- -summary

-all

```
-all          Displays help information for all available
              commands and topics. When used with the -brief
              option, displays only synopsis information. When
              used without any other options or arguments,
              displays a list of available commands (same as
              specifying the help command without any options or
              specifying the -summary option).

              Note: When you use the -all option and specify one or
              more topic names, the entire help file (full
              documentation on all commands and topics) is
              displayed. Because of the size of the help file,
              this operation may take a while to complete.
```

-brief

```
-brief          Displays the synopsis information for each specified
                topic. The synopsis for individual DesignSync
                commands is typically the command usage, while for
                other topics, it is a brief topic summary. If you do
                not specify one or more topics, brief help is
                displayed for all commands.
```

-output

```
-output <file> This option is used to write help topics to a text
               file instead of displaying them. When used with the
               -all option, a file is created containing all the
               available topics. This can be combined with the
               -brief option to provide a full synopsis of all
               topics.

               Caution: If the file specified already exists,
               its contents will be erased.
```

-summary

```
-summary        Displays the list of available help
                topics. This option is the same as specifying
                the help command without any options or arguments.
```

RETURN VALUE

```
none
```

EXAMPLES

```
The following example returns brief (synopsis) information for
the 'ci' and 'co' commands:
  dss> help -brief ci co

The following example returns help information for the 'url vault'
command. The double quotes are required, and there must be exactly
one space between 'url' and 'vault':
  dss> help "url vault"

You can get the same help information by using the command's -help or -?
option:
```

```
  dss> url vault -help
or
  dss> url vault -?
```

locate

locate Command

NAME

```
locate              - Finds a specified object on the search paths
```

DESCRIPTION

```
This command searches the Synchronicity paths for a specified
object, either a file or directory. You can find either the first
occurrence of the object (the default) or all occurrences of the
object.

On the server side, the following paths are searched in this order:

 <SYNC_CUSTOM_DIR>/servers/<host>/<port>
 <SYNC_CUSTOM_DIR>/site
 <SYNC_CUSTOM_DIR>/enterprise
 <SYNC_DIR>

On the client side, the following paths are searched in this order:

 <SYNC_USER_CFGDIR>
 <SYNC_SITE_CUSTOM>
 <SYNC_ENT_CUSTOM>
 <SYNC_DIR>

The environment variables match the following paths:
Variable name:             Path:
--------------             -----
SYNC_DIR                   Synchronicity installation directory
SYNC_CUSTOM_DIR            <SYNC_DIR>/custom.
SYNC_SITE_CUSTOM           <SYNC_CUSTOM_DIR>/site.
SYNC_ENT_CUSTOM            <SYNC_CUSTOM_DIR>/enterprise
SYNC_USER_CFGDIR           User-specific customization files
                            (UNIX default <HOME>/.synchronicity)
                            (Windows default %AppData%\Synchronicity)

You cannot specify path names containing the ".." relative path
notation. If you try to include this notation, the locate command
throws an exception.

On the client side, you must run this command in stcl/stclc mode.
```

SYNOPSIS

```
locate [-env | -path | -url] [-first | -all] [-nothrow] [-reverse]
       [--] <ObjectName>
```

ARGUMENTS

- Object Name

Object Name

ObjectName	The name of the file or directory you want to locate.

OPTIONS

- -all
- -env
- -first
- -nothrow
- -path
- -reverse
- -url
- --

-all

-all	Returns all occurrences of ObjectName in the search paths. The default behavior is -first.

-env

-env	Returns environment variables in place of literal path names. For example, instead of returning: /home/john/syncinc/custom/site/share/tcl/test.txt this command returns: <SYNC_CUSTOM_DIR>/site/share/tcl/test.txt The -env, -path, and -url options are mutually exclusive; -path is the default.

-first

-first — Returns the first occurrence of ObjectName in the search paths. This behavior is the default.

-nothrow

-nothrow — If the object is not found on the search paths, returns an empty string. Without this option, the locate file command throws an exception when the search object is not found.

-path

-path — Returns the full file system path of the object. The -env, -path, and -url options are mutually exclusive; -path is the default.

-reverse

-reverse — When used with the -all option, returns the path in reverse search order. You get an error if you try to use this option without the -all option.

-url

-url — Returns the server URL, prepended by:

- Server area: /syncserver
- Custom area: /syncsite
- Enterprise area: /syncent
- Installation area: /syncinc

If you use the server-side -url option, specify the path to the search object relative to the one of the share/content directories:

<SYNC_CUSTOM_DIR>/servers/<host>/<port>/share/content
<SYNC_CUSTOM_DIR>/site/share/content
<SYNC_CUSTOM_DIR>/enterprise/share/content
<SYNC_DIR>/share/content

This option is only available when run from the server. The -env, -path, and -url options are mutually exclusive; -path is the default.

--

--	Indicates that the command should stop looking for command options. Use this option when you specify an object whose name begins with a hyphen (-).

RETURN VALUE

When you use the -first option, returns a string indicating where the specified object was first located in the search path. When you use the -all option, returns a list of the places where the specified object was found in the search paths.

The format of the path returned depends on the options you use.

EXAMPLES

Examples of using locate

The following example searches for a file called test.txt. A user has put copies of the file in two of the directories on the search path (<SYNC_CUSTOM_DIR>/site/test.txt and <SYNC_DIR>/test.txt).

The command

```
locate -env test.txt
```

returns the first location found on the search path:

```
$SYNC_CUSTOM_DIR/site/test.txt
```

When used with the -all option, the command finds the file in two of the directories on the search path.

```
{$SYNC_CUSTOM_DIR/site/test.txt} {$SYNC_DIR/test.txt}
```

In both cases, the -env option causes the paths to be displayed using environment variables.

Example of Using -nothrow with locate

The following example uses the same problem as the previous example, but includes the use of the -nothrow option to avoid throwing an exception if the file is not found. Without the -nothrow option, you need to write Tcl code to deal with exceptions. For example:

```
#Trying to find file
if [catch {set filename [locate share/test.txt]} msg] {
  puts "The file was not found on the search path. The result is $msg"
}

The -nothrow option lets you write simpler and less error-prone
code that gets the same result:

#Trying to find file
set filename [locate -nothrow share/tcl/test.txt]
if {$filename == ""} {
  puts "The file was not found on the search path.\n"
}
```

ls

ls Command

NAME

```
ls                  - Lists information about the specified objects
```

DESCRIPTION


```
This command lists information about the specified objects. You
typically specify objects such as folders, files, and
collection objects such as Cadence cell views and CTP collections.

Note: The ls command reports revision control information about a
collection member as if it were the collection itself. In other
words, if a collection is locked and has version 1.1, then that
information appears in the ls output for all of the collection's
members. The only exception is when the collection is modified; in
this case the ls command shows which members are modified or new.

With the ls command you can also specify server-side DesignSync
objects such as vaults, branches, and versions; however, the 'ls'
```

command is optimized to give you quick information about workspace objects. By default, 'ls' reports information accessed only from local metadata, although you can choose to view server information as well. (See "Report Options" below for details). You cannot view server-side module objects with ls.

If you specify a container object -- an object that contains other objects, such as folders and vaults -- 'ls' returns information about the container object's contents.

You can use wildcards to specify objects to be listed. If you do not specify an argument or the -selected option, then the default is to list the contents of the current folder.

All mirror directories can be treated in the same way as your DesignSync work areas.

This command is subject to access controls on the server. See the ENOVIA Synchronicity Access Control Guide for details.

This command supports the command defaults system.

Notes for Module Objects and Module Snapshots (Module-based)

The ls command does not provide a module manifest. If you specify a workspace module, module instance, or module directory, ls provides information about the module members contained in the workspace. For example, the -merged option allows you to view the status of object in your workspace after a module merge has been performed.

When ls includes a workspace that has been populated with a blend of a main selector and a module snapshot, the objects in the workspace are compared against both the module snapshot and the main selector. Therefore status of a member populated from the module snapshot is calculated against the module snapshot branch version, and the status of a member populated from the main selector is calculated against the corresponding vault version.

The 'ls' command does not follow soft links such as module caches (Mcaches). In cases where softlinks exist, 'ls' ignores them and only lists only objects physically present in the directory structure.

The -nomodulerecusive option, which allowed you to list a directory without including module contents, is no longer supported. To list a directory and omit contents, use the exclude filter.

Use the 'compare' command to compare objects in workspaces.

Notes on Legacy Modules (Legacy-based)

If the container is a folder containing a legacy module configuration, 'ls' follows its hierarchical references if you specify the -recursive option. If the container is a vault, 'ls' ignores the hierarchical references. The 'ls' command does not follow DesignSync configuration REFERENCEs. In these cases, 'ls' lists only objects physically present in the directory structure.

The hcm showconfs command can be used to get information about legacy module configurations.

If you are actively developing with legacy modules and have legacy module mode enabled, use the hcm show* commands to list information about legacy modules.

Note: The -nomodulerecusive option, which allowed you to list a directory without including module contents, is no longer supported. To list a directory and omit contents, use the exclude filter.

Notes for Files-Based Objects (File-based)

To list objects in vaults according to criteria such as a configuration name, use the 'contents' command. Use the 'compare' command to compare objects in vaults, configurations, and workspaces.

Report Options

The -report option lets you specify what information 'ls' returns. You can specify one of the predefined modes (silent, brief, normal, verbose, status), or you can specify one or more data keys to specify exactly the information you want. For example, if you want to see objects' names, fetched times, and last-modified times, specify '-report NFM'.
Note that:

- All data keys must be uppercase.
- The object's name is always included in the listing whether or not you specify the 'N' key.
- If you specify an unused key (such as 'E'or 'Q'), it is ignored.
- Some keys return values on a single line, while others can span multiple lines. Single-line values are always displayed first, followed by multi-line output.

Note: You can add or remove keys from the predefined modes using the + and - keys with the report data keys listed below. For example, report -normal specifies the MDVLN data keys. If instead of specifying D, workspace status, you chose to specify S, status, you could type 'ls -report normal-D+S' (or '-report MSVLN').

Tip: Because report -normal is the default, 'ls -report normal-D+S' could also be specified as 'ls -report -D+S'; the normal option is implied.

The predefined modes are defined as follows:

Mode	Data Key Definition
brief	N
normal	MDVLN
status	MSRUN
verbose	OMSRUNCTAX
silent	!

The default behavior if -report is not specified is '-report normal'. The 'normal' data keys (MDVLN) generate a quick listing because these keys do not access the server vault. To view more detailed status including revision control status and the username of the locker, specify '-report status'. The 'status' report accesses the server vault and therefore is typically not as fast as the 'normal' report.

Note: The '-report normal' command shows the fetched version rather than the upcoming version for locked or auto-branched objects. Use '-report status' which includes the 'R' report key to show the upcoming version instead.

Report Data Keys Table (Module-based)

The following table lists the -report data keys. The table indicates whether the data key accesses the server vault or gathers the data locally and thus can provide a quicker listing. The table also provides the property name corresponding to each data key.

Note: When a * appears at end of a description, it indicates that the data key uses the current module version on the selected branch. Otherwise, the command displays the information for version populated into the workspace.

Data Key	Property Name	From Vault?	# of Lines	Description
!	N/A	N/A	N/A	Silent output. Use the '!' key with no other data keys to suppress the listing, although errors are still displayed. If the '!' key is used with other data keys, it is ignored. This key is useful with the -addselect option when you are only interested in defining your select list.
A	branchtags	Yes	Multi	Branch tags. Use 'H' for single-line format.* Note: Module members cannot be tagged. The displayed tag applies to the module.
C	comments	Yes	Multi	Original and checkout log comments.

				Original Log comments include: o author o creation time of the current version o check-in comments, if any. Checkout Log comments, if any, include: o check-out comments o changes made from the Revision Log field on the DesignSync GUI (File=>Properties=>Revision Control).*
D	wsstatus	No	Single	Workspace status. These options are explained more fully in the status values table. Values include: o Absent - for objects fetched but no longer present. o Added - for module members added to a module, but not yet checked in. o Added By Merge/Needs Checkin o Locally Modified - for objects that have been modified in the workspace. o Moved - workspace object has moved. o Null - for objects in the same state as when the directory was last updated. o Out-of-sync - when a module member version is incorrect for the module version in the workspace. o Remove for Merge o Removed - workspace module member has been marked for removal. o Rename for Merge o Unresolved Conflicts Workspace status is a subset of the 'S' revision control status key, displaying status information available from the workspace, however some status information is not reported by the 'S' report option, so you might need to specify both 'S' and 'D' for complete status.
F	fetchtime	No	Single	Fetched time. Note: If you used 'populate -mirror' to fetch the object to your work area, then the fetchtime for the object is listed as 0.
G	tags	Yes	Single	Version tags. Use 'T' for multi-line

				format.* Note: Module members cannot be tagged. The displayed tag applies to the module.
H	branchtags	Yes	Single	Branch tags. Use 'A' for multi-line format.* Note: Module members cannot be tagged. The displayed tag applies to the module.
I	uid	No	Single	Object UID.
L	fetchedstate	No	Single	Fetched state. Options include: o Copy o Lock o Reference - for unlocked references o Cache o Checkin Excluded - unmanaged object, excluded by exclude file. o Null (" ")- for unmanaged or non-versionable objects. The fetched state displays with header "Type" in the report table. Note: The fetched state for Locked references is "Lock", not "Reference". Use the 'V' mode, which displays "Refers to" for references, with 'L' to determine whether an object is a locked reference.
M	mtime	No	Single	Last-modified time. For a listing of vault locks, shows lock time. For references, this field is empty (" ").
N	name	No	Single	Name and, if -path or -fullpath is specified, the path. Note that objects' names are displayed even when 'N' is not specified.
O	type	No	Single	Object Type: For modules members and other DesignSync objects, the types are: o File, o Folder o Project o Absent File (a locked reference or deleted file) o Referenced File o Link to File o Link to Folder o Link to Mcache o Cached File o Vendor-specific object types such as Cadence or Synopsis libraries,cells, and cell views, or CTP collection object types. Note: When running the ls-foreach function, the property type name

				used is otype.
P	selector	No	Single	Persistent selector list (as defined by the "setselector" command, or "Trunk" by default). Note: If a folder is a member of more than one module, the selector displays as an empty ("") value.*
Q	csum	No	Single	The checksum of the object. If the object is not in source control, the checksum is 0.
R	upcoming	Yes	Single	Current version, and upcoming version if object is locked or auto-branched. For a locked reference, 'R' shows the current version and upcoming version to which it refers. Note: upcoming versions are not applicable for module members. The 'R' option only provides the current version (equivalent to 'V').
S	status	Yes	Single	Server status. These options are explained more fully in the status values table. Values include: o Up-to-date o Needs Merge o Needs Update o Added By Merge/Needs Checkin o Added o Absent - indicates a locked reference or a file deleted from the operating system o Unknown. See the Status Value table below for descriptions of these values.
T	tags	Yes	Multi	Version tags. Use 'G' for single-line format.* Note: Module members cannot be tagged. The displayed tag applies to the module.
U	user	Yes	Single	Username of the locker, or empty if the object is not locked. If the object is locked in this location, the report displays an asterisk (*) after the username. Modules members display as locked when the member is locked.
V	version	No	Single	Fetched version. Display options include: o Version number o Unmanaged - for an object with no local metadata. For example, recreated files display as Unmanaged, because their metadata was removed by a previous rmfile. o Null ("") - for non-versionable objects. Note: This does not show the

				upcoming version. To show the upcoming version for an object, use the 'R' option. Module members do not have upcoming versions.
W	objtype	No	Single	Web object types include: o Folder, o File, o Project, o Link to Folder. o Link to Mcache
X	owner	No	Single	Owner of the object. For collections - the collection to which a collection member belongs. For modules - the module to which a module member belongs. For a folder - all the modules that own the folder. Note: If ls is restricted to a single module, using the -modulecontext option, the specified module is the only owner shown.
Y	memberof	No	Single	Module instance name. If the object(s) is not a module member, this field is blank.
Z	size	No	Single	Size of the object in bytes. For collection objects, this option displays either the total number of member files in the collection, or the size of the objects in bytes. For more information on choosing the display value for Z, see the SyncAdmin help file.
	error	Yes	Multi	Error message. If ls is unable to fetch data for an object, the 'error' property automatically displays with the error.

Status Values for Modules and Modules Members (Module-based)

The following table describes the status values. Server status values are specified using the 'S' key or the '-report status' option. Workspace status values are specified using the -'D' key, or the '-report normal' option. For ease of use, this list is in alphabetical order.

Status Value	Description
Added	Indicates that the object has been added to the module, but has not been checked in.
Added By merge, Needs Checkin	Indicates that the file was introduced to the work area by a merge or overlay operation and does not exist on the current branch. These objects do not

	show as modified by the ls command. Note: If a file was added by merge, but is no longer present in the workspace, it will display with a status of "Added by merge, Needs Checkin", not a status of "Absent".
Absent	Indicates an object that is unexpectedly absent from the workspace. Typically an object is listed as "Absent" if it was deleted from the workspace using operating system commands, leaving behind the local metadata. Note: If a file was added by merge, but is no longer present in the workspace, it will display with a status of "Added by merge, Needs Checkin", not a status of "Absent".
Locally Modified	Indicates that the file has been edited since it was fetched and a more recent version has not been checked into the branch, or that an add has been performed on a module member that has not been checked into the module.
Locally Modified, Moved	Indicates that a module member has been modified, the location of the module member has changed, and the modified, moved version has not been checked into the branch.
Moved	Indicates that a module member with no content changes has been moved or renamed in the workspace, and the moved version has not been checked into the branch. Note: If a module member was both moved and removed, it is displayed only as removed.
Needs Merge [<change>]	Indicates that a file has been locally modified and the version is not correct for the current selector. For modules, an additional value indicating the type of change may appear in brackets [] after the Needs Merge status value: o <Version number> - the version of the member in the module version. o Moved - the module member has a different natural path than the one expected by the module version. o Removed - the module member is not in the module version. o <Version number>,Moved - the module member is both a different version and located at a different natural path than the module version.
Needs Update [<change>]	Indicates that you have an incorrect version on the given branch. For modules, an additional value indicating the type of change may appear in brackets after the

	Needs Update status value: o <Version number> - the version of the member in the module version. o Moved - the module member has a different natural path than the one expected by the module version. o Removed - the module member is not in the module version. o <Version number>,Moved - the module member is both a different version and located at a different natural path than the module version.
Out-of-sync	Indicates the version of the file in the workspace is out of sync with the expected module version. Note: This value is part of the workspace status, the '-D' key must be specified to determine if the workspace contains out of sync objects.
Remove for Merge	Indicates that an object present in the workspace was removedon the branch being merged into the workspace. To remove the file on the server in the current branch, remove this file using the remove command for module members or the retire command for non-module members. This is a workspace status value. Note: Objects that were removed with rmvault are no longer present on the DesignSync server and do not show up in any ls query.
Removed	Indicates that a module member has been removed from the workspace and has not been checked into the branch. The module member is not marked as Absent, and information about the last fetched version of the module member is displayed. Notes: o If a module member was renamed and subsequently removed before a checkin operation was performed to update the server version, ls reports that object only as Removed. o The type column is not maintained for removed items.
Remove for Merge	Indicates that an object present in the workspace was removed, being retired on the branch being merged into the workspace. To remove the file on the server in the current branch, remove this file using the retire command. This is a workspace status value.
Rename for Merge Merge branch path: <naturalpath>	Indicates that an object present in the workspace was renamed to the <naturalpath> value on the branch being merged into the workspace. To incorporate this change into the current branch, move this file to the naturalpath name using mvmember. This status is only applicable to cross-branch merge operations. This is a workspace status value.

	Note: If this file has any other status information, such as Absent or modified, the status column shows all the appropriate values separated by a comma.
Unknown	Indicates that the version of the file in the workspace cannot be determined from the local metadata.
Unresolved Conflicts	Indicates that a merge of version contents resulted in conflicts. Any object marked as containing unresolved conflicts is considered locally modified.
Up-to-date	Indicates that the module member version matches the correct version for the module selector.

Report Data Keys Table (File-based)

The following table lists the -report data keys. The table indicates whether the data key accesses the server vault or gathers the data locally and thus can provide a quicker listing. The table also provides the property name corresponding to each data key.

Data Key	Property Name	From Vault?	# of Lines	Description
!	N/A	N/A	N/A	Silent output. Use the '!' key with no other data keys to suppress the listing, although errors are still displayed. If the '!' key is used with other data keys, it is ignored. This key is useful with the -addselect option when you are only interested in defining your select list.
A	branchtags	Yes	Multi	Branch tags. Use 'H' for single-line format.
C	comments	Yes	Multi	Original and checkout log comments. Original Log comments include: o author o creation time of the current version o check-in comments, if any. Checkout Log comments, if any, include: o check-out comments o changes made from the Revision Log field on the DesignSync GUI (File=>Properties=>Revision Control).

D	wsstatus	No	Single	Workspace status. These options are explained more fully in the status values table. Values include: o Absent - for objects fetched but no longer present. o Locally Modified - for objects that have been modified in the workspace. o Moved - workspace object has moved. o Null - for objects in the same state as when the directory was last updated. o Unresolved Conflicts Workspace status is a subset of the 'S' revision control status key, displaying status information available from the workspace, however some status information is not reported by the 'S' report option, so you might need to specify both 'S' and 'D' for complete status.
F	fetchtime	No	Single	Fetched time. Note: If you used 'populate -mirror' to fetch the object to your work area, then the fetchtime for the object is listed as 0.
G	tags	Yes	Single	Version tags. Use 'T' for multi-line format.
H	branchtags	Yes	Single	Branch tags. Use 'A' for multi-line format.
I	uid	No	Single	Object UID.
L	fetchedstate	No	Single	Fetched state. Options include: o Copy o Lock o Mirror o Reference - for unlocked references o Cache o Checkin Excluded - unmanaged object, excluded by exclude file. o Null (" ")- for unmanaged or non-versionable objects. The fetched state displays with header "Type" in the report table. Note: The fetched state for Locked references is "Lock", not "Reference". Use the 'V' mode, which displays "Refers to" for references, with 'L' to determine whether an object is a locked reference.

M	mtime	No	Single	Last-modified time. For a listing of vault locks, shows lock time. For references, this field is empty (" ").
N	name	No	Single	Name and, if -path or -fullpath is specified, the path. Note that objects' names are displayed even when 'N' is not specified.
O	type	No	Single	Object Type: o File, o Folder o Project o Absent File (a locked reference or deleted file) o Referenced File o Link to File o Link to Folder o Cached File o Mirrored File o Vendor-specific object types such as Cadence and Synopsys libraries, cells, and cell views, or CTP collection object types. For vault objects, types include: o File o Version o Branch Point Version. For non-versionable objects: o Non-versionable. Note: When running the ls-foreach function, the property type name used is otype.
P	selector	No	Single	Persistent selector list (as defined by the "setselector" command, or "Trunk" by default).
Q	csum	No	Single	The checksum of the object. If the object is not in source control, the checksum is 0.
R	upcoming	Yes	Single	Current version, and upcoming version if object is locked or auto-branched. For a locked reference, 'R' shows the current version and upcoming version to which it refers.
S	status	Yes	Single	Server status. These options are explained more fully in the status values table. Values include: o Up-to-date o Needs Merge o Needs Update o Absent - indicates a locked reference or a file deleted from the operating system o Unknown. The status is preceded by [Retired] if the current branch is retired. See the Status Value table below

				for descriptions of these values.
T	tags	Yes	Multi	Version tags. Use 'G' for single-line format.
U	user	Yes	Single	Username of the locker, or empty if the object is not locked. If the object is locked in this location, the report displays an asterisk (*) after the username.
V	version	No	Single	Fetched version. Display options include: o Version number o Unmanaged - for an object with no local metadata. For example, recreated files display as Unmanaged, because their metadata was removed by a previous rmfile. o Null ("") - for non-versionable objects. Note: This does not show the upcoming version. To show the upcoming version for an object, use the 'R' option.
W	objtype	No	Single	Web object types include: o Folder, o File, o Project, o Link to Folder. For vault web objects, object types include: o File, o Version o Branch Point Version.
X	owner	No	Single	Owner of the object. For collections - the collection to which a collection member belongs. For a folder - all the modules that own the folder.
Y	memberof	No	Single	This field is blank.
Z	size	No	Single	Size of the object in bytes. For collection objects, this option displays either the total number of member files in the collection, or the size of the objects in bytes. For more information on choosing the display value for Z, see the SyncAdmin help file.
	error	Yes	Multi	Error message. If ls is unable to fetch data for an object, the 'error' property automatically displays with the error.

Status Values for File-Based Objects (File-based)

The following table describes the status values. Server status values are

specified using the 'S' key or the '-report status' option.
Workspace status values are specified using the -'D' key, or the '-report normal' option. For ease of use, this list is in alphabetical order.

Status Value	Description
Added By merge, Needs Checkin	Indicates that the file was introduced to the work area by a merge or overlay operation and does not exist on the current branch. These objects do not show as modified by the ls command. Note: If a file was added by merge, but is no longer present in the workspace, it will display with a status of "Added by merge, Needs Checkin", not a status of "Absent".
Absent	Indicates an object that is unexpectedly absent from the workspace. Typically an object is listed as "Absent" if it was deleted from the workspace using operating system commands, leaving behind the local metadata. Note: If a file was added by merge, but is no longer present in the workspace, it will display with a status of "Added by merge, Needs Checkin", not a status of "Absent".
Locally Modified	Indicates that the file has been edited since it was fetched and a more recent version has not been checked into the branch, or that an add has been performed on a module member that has not been checked into the module.
Needs Merge [<change>]	Indicates that a file has been locally modified and the version is not correct for the current selector.
Needs Update [<change>]	Indicates that you have an incorrect version on the given branch.
Remove for Merge	Indicates that an object present in the workspace was removed, being retired on the branch being merged into the workspace. To remove the file on the server in the current branch, remove this file using the retire command. This is a workspace status value. Note: Objects that were removed with rmvault are no longer present on the DesignSync server and do not show up in any ls query.
Unknown	Indicates that the version of the file in the workspace cannot be determined from the local metadata.
Unresolved Conflicts	Indicates that a merge of version contents resulted in conflicts. Any object marked as containing unresolved conflicts is considered locally modified.

Up-to-date	Indicates that the file is currently the correct version for the selector.
[Retired]	Indicates that the current branch is retired. This is not a state of its own; rather it is a prefix to one of the other states. For example, the status column might contain: [Retired] Locally Modified Because the [Retired] status is a prefix to other status terms, sorting causes retired items to be grouped either at the beginning or end of the listing, independent of the items' local state. Note: Retired is not a valid state for module members.

SYNOPSIS

```
ls [-[no]addselect] [-[un]changed] [-exclude <string>]
   [-filter <string>] [-format list | text] [-[no]header]
   [-hreffilter <string>] [-[un]locked] [-[un]modified]
   [-modulecontext <context>] [-[no]needsmerge [-branch <branch>]]
   [-[un]managed] [-merged added|rename|remove|all|""]
   [-output <file> | -stream <stream>] [-[no]path | -fullpath]
   [-[no]recursive] [-report <mode>[+<mode>][-<mode>]]
   [-[no]selected] [-[non]versionable] [-workspace | -vault>]
   [-writableunlocked] [-xtras <xtras>] [--] [<argument>
   [<argument>...]]
```

ARGUMENT

- Server Folder
- Server Object
- Workspace Module (Module-based)
- Module Member or Folder (Module-based)
- External Module (Module-based)
- DesignSync Object or Unmanaged Objects (File-based)

Server Folder

<server folder>	Provides information about the specified object on the server, and if you specify the -recursive option, all subfolders. Specify the object with the sync URL in the format: sync://<host>:<port>/<path>/<folder>

Server Object

<server object>	Provides information about the specified object on the server. Specify the object with the sync URL in the format: sync://<host>:<port>/<path>/<object>

Workspace Module (Module-based)

<workspace module>	Provides information about the files in the specified module and the hierarchical references in the specified module. To view information about the module itself, use the "show" commands: hcm showconfs, showhrefs, showmcache, showmods, and showstatus. To view a list of locked module elements, use the showlocks command.

Module Member or Folder (Module-based)

<module member | module folder>	Provides information about the specified module members, and if you specify the -recursive option, all subfolders. The -modulecontext option restricts the list to objects that are members of the specified module.

External Module (Module-based)

<external module>	Specifies the module instance of the external module in the workspace or the URL of the external module version to which you wish to create the connection. An external module is an object or set of objects managed by a different code management system but available for viewing and integration through DesignSync. Specify the external module in the workspace as a module instance name. Specify the external module Server URL as follows: sync[s]:///ExternalModule/<external-type>/<external-data> where ExternalModule is a constant that identifies this URL as an external module URL, <external-type> is the name of the external module procedure, and <external-data> contains the parameters and options to pass to the procedure. These parameters and options can be passed from the procedure to the external code management system or to DesignSync. Note: In order to specify an external module, you must have previously populated the module in the workspace. You may also specify the external module by href name only.

DesignSync Object or Unmanaged Objects (File-based)

<DesignSync object | DesignSync folder | unmanaged object | unmanaged folder>	Provides information about the specified DesignSync object or folder. If you specify a folder, you can use the -recursive option to get information about all subfolders contained in the specified folder.

OPTIONS

-[no]addselect

-[no]addselect	Indicates whether objects matching the ls specification should be added to the select list. -noaddselect does not add the objects displayed by ls to the select list. (Default) -addselect adds the objects that match your 'ls' specification to your select list. You can use this option in conjunction with '-report !' to suppress 'ls' output if you only want to update your select list.

-branch

-branch <branch>	Use the -branch option with the -needsmerge or -noneedsmerge option to compare objects against the specified branch rather than against the current branch. Specifying the -branch option without the -needsmerge or -noneedsmerge option generates an error. Specifying -branch with -changed or -unchanged generates an error, as the -changed option only compares objects against the current branch. The -branch option accepts a branch or version tag or a branch numeric. It does not accept a selector or selector list. If <branch> resolves to a version, the branch of that version is used for the comparison.

-[un]changed

-[un]changed	Determines whether to show only objects that are identical (up-to-date) in both the vault and in the workspace, or only objects with different versions in the vault and the workspace. Objects can have different version in the vault or workspace if local modifications are made or if there is a newer version on the server than the last version fetched. -unchanged shows only objects that are up-to-date. -changed shows only objects that are not up-to-date. An object is considered changed if it is locally modified, if there is a newer version in the vault, or if there's a structural change to a module, such as moved or removed module members. The -changed option is a combination of the -unmanaged and -modified options and the "Needs Update" state. To show objects that are

```
                    locally modified without checking whether there
                    are newer versions in the vault, use the (faster)
                    -modified option. Unmanaged objects or module
                    members that have been added, but not checked in
                    are always considered changed.

                    Specifying both -changed and -unchanged is
                    equivalent to specifying neither option: 'ls'
                    displays both changed and unchanged objects. If
                    -changed is specified with either -modified or
                    -needsmerge or -unchanged is specified with
                    either -unmodified -noneedsmerge, only the
                    -[un]changed option is processed, because the
                    changes include both merge information and
                    modified information.  If -changed is specified
                    with either -unmodified or -noneedsmerge, or
                    -unchanged is specified with either -modified or
                    -needsmerge , ls returns an error, as these
                    options are mutually exclusive.

                    The -[un]changed options only apply to
                    workspaces. They are silently ignored for 'ls'
                    of vault objects.
```

-exclude

```
 -exclude <string>  Specifies a glob-style expression to exclude
                    matching object names from the listing. The string
                    you specify must match the name of the object as
                    it would have appeared in the listing. Generally,
                    you can specify the leaf name of the objects.  If
                    you use the -fullpath option, you must specify
                    a glob expression that matches the full path, for
                    example, file:///home/karen/Projects/Asic/test.v.
                    If you use the -path option, you must specify a
                    glob expression that matches the relative path,
                    for example, ../top*.

                    Important: The exclude option is applied after the
                    -filter option and is used to further refine the
                    filter.

                    By default, the 'ls' command does not exclude
                    the objects in the global exclude lists
                    (set using Tools->Options->General->Exclude
                    Lists or using SyncAdmin:General->Exclude Lists).
                    To exclude these objects from an 'ls' listing,
                    apply the -exclude option with a null string:
                       ls -exclude ""
                    The objects in the global exclude lists are
                    appended to the 'ls' exclude list if you
                    exclude other values:
                       ls -exclude "README.txt"
```

-filter (Module-based)

-filter <string>

Specify one or more extended glob-style expressions to identify an exact subset of module objects on which to operate. Use the -exclude option to filter out DesignSync objects that are not module objects.

The -filter option takes a list of expressions separated by commas, for example: -filter +top*/.../*.v,-.../a*

Prepend a '-' character to a glob-style expression to identify objects to be excluded (the default). Prepend a '+' character to a glob-style expression to identify objects to be included. Note that if the list of expressions begins with an include character ('+'), the filter excludes all objects except those that match the include string.

Specify the paths in your glob-style expressions relative to the current directory, because DesignSync matches your expressions relative to that directory. For submodules followed through hrefs, DesignSync matches your expressions against the objects' natural paths, their full relative paths. For example, if a module, Chip, references a submodule, CPU, and CPU contains a file, '/libs/cpu/cdsinfo.tag', DesignSync matches against '/libs/cpu/cdsinfo.tag', rather than matching directly within the 'cpu' directory.

If your design contains symbolic links that are under revision control, DesignSync matches against the source path of the link rather than the dereferenced path. For example, if a symbolic link exists from 'tmp.txt' to 'tmp2.txt', DesignSync matches against 'tmp.txt'. Similarly for hierarchical operations, DesignSync matches against the unresolved path. If, for example, a symbolic link exists from dirA to dirB, and dirB contains 'tmp.txt', DesignSync matches against 'dirA/tmp.txt'.

The extended glob-style expressions you use to filter the objects are standard glob-style expressions, but they are extended so that you can use the "..." syntax to indicate that the expression matches any number of directory levels. For example, the expression, "top/.../lib/*.v" matches *.v files in a directory path that begin with "top", followed by zero or more levels, with one of those levels containing a lib directory. The command traverses the directory structure. If a directory name

matches an exclude clause of the filter, then the entire directory and all its contents are filtered (the command stops descending at that point), otherwise the command continues traversing the directory structure searching for matching objects.

The -filter option does not override the exclude list set using SyncAdmin's General=>Exclude Lists tab or with the -exclude command line option; the items in the exclude list are combined with the filter expression. For example, an exclude list of "*%,*.reg" combined with '-filter .../*.doc' is equivalent to: '-filter .../*.doc,.../*%,.../*.reg'.

-format (Module-based)

-format

Specifies whether the format of the 'ls' output should be a Tcl list or formatted text:

list Display a list with the following format:

```
{
 name <name>
 type file | folder | version | branch
 props <prop_list>
 objects <object_list>
}
```

For a list of properties, see the "Report Options" table above. Container objects, including folders and branch-point versions, have an 'objects' list containing their subcomponents. The list is the return value of the 'ls' command and is echoed to the screen by the dss/stcl shells. If '-format list' is used with the '-output' or '-stream' option, a formatted list is generated in the file or stream.

Note: The type for module objects is 'module'. The module folder type is folder. If a module is listed recursively, an addition module property is added to the results for each module referenced in the hierarchy.

The module type entry includes the following information:

- o URL of the module
- o fetched version of the module
- o module base directory
- o relative path to the module from the top-level module

```
                The ls command does not operate on
                Module branches or versions.

                To process the results, use the
                ls-foreach function.

        text    Display a text table with headers and
                columns. (Default)  Objects shown in
                alphabetical order.  If 'format text' is
                used, 'ls' has no return value, but 'ls'
                prints the text table to the screen.

        Notes:
        o If an error occurs while accessing an object's
          vault data, the text output prints an error
          message preceding the object. For list output,
          the message precedes the list and the object's
          property list includes an 'error' property
          containing the error message. In both cases,
          the object's revision control status is
          'Unknown'.
        o If '-format' is used with '-report silent' or
          '-report !', the silent option overrides
          the '-format' option and the list or text
          output is suppressed.
        o For recursive listings, if multiple objects
          have the same leaf name, use the -path or
          -fullpath option to differentiate between
          these objects.
```

-format (File-based)

```
 -format        Specifies whether the format of the 'ls' output
                should be a Tcl list or formatted text:

                list    Display a list with the following format:
                          {
                           name <name>
                           type file | folder | version | branch
                           props <prop_list>
                           objects <object_list>
                          }

                        For a list of properties, see the
                        "Report Options" table above.  Container
                        objects, including folders and branch-point
                        versions, have an 'objects' list containing
                        their subcomponents.  The list is the return
                        value of the 'ls' command and is echoed to
                        the screen by the dss/stcl shells.  If
                        '-format list' is used with the '-output'
                        or '-stream' option, a formatted list is
                        generated in the file or stream.
```

```
                The ls command does not operate on
                Module branches or versions.

                To process the results, use the
                ls-foreach function.

        text    Display a text table with headers and
                columns. (Default)  Objects shown in
                alphabetical order.  If 'format text' is
                used, 'ls' has no return value, but 'ls'
                prints the text table to the screen.

        Notes:
        o If an error occurs while accessing an object's
          vault data, the text output prints an error
          message preceding the object. For list output,
          the message precedes the list and the object's
          property list includes an 'error' property
          containing the error message. In both cases,
          the object's revision control status is
          'Unknown'.
        o If '-format' is used with '-report silent' or
          '-report !', the silent option overrides
          the '-format' option and the list or text
          output is suppressed.
        o For recursive listings, if multiple objects
          have the same leaf name, use the -path or
          -fullpath option to differentiate between
          these objects.
```

-fullpath

```
 -fullpath          Display object names as full URLs. By default,
                    only the object name is displayed.
                    The -path and -fullpath options are mutually
                    exclusive.
```

-[no]header

```
 -[no]header        Indicates whether the command should display with
                    field headers before each column in the output.

                    -noheader does not display the fielder header.

                    -header displays a field header at the top of the
                    'ls' output.(Default)
```

-hreffilter (Module-based)

-hreffilter <string>	Excludes href values during recursive operations on module hierarchies. Because hrefs link to submodules, you use -hreffilter to exclude particular submodules. The hreffilter value is matched against both the name of the href and the target module name. Note that unlike the -filter option which lets you include and exclude items, the -hreffilter option only excludes hrefs and, thus, their corresponding submodules. Specify the -hreffilter string as a glob-style expression. The string must represent a simple leaf name; you cannot specify a path. DesignSync matches the specified href filter against hrefs anywhere in the hierarchy. Thus, DesignSync excludes all hrefs by this leaf name; you cannot exclude a unique instance of the href. You can prepend the '-' exclude character to your string, but it is not required. Because the -hreffilter option only supports excluding hrefs, a '+' character is interpreted as part of the glob expression.

-[un]locked (Module-based)

-[un]locked [-workspace| -vault]	Determines whether to display only objects that are locked or objects that are not locked. Specifying the -workspace or -vault option allows you to further restrict the ls output to searching in only the local workspace or searching only on the server. Specifying -workspace provides a faster response to time because the 'ls' command accesses only the local metadata and not the server data. -unlocked shows only objects that are currently unlocked. -locked shows only objects that are currently locked by any user. You can differentiate between objects locked by you and others by noting the fetched state (shown with header "Type"). If you have a lock on the object, the fetched state is "Lock". If a module branch is locked, all module members returned by ls will display as locked. Note: Use the showlocks command to get information about server-side Module locks. Specifying both -locked and -unlocked is equivalent to specifying neither option: 'ls' displays both locked and unlocked objects.

-[un]locked (File-based)

-[un]locked [-workspace| -vault]	Determines whether to display only objects that are locked or objects that are not locked. Specifying the -workspace or -vault option allows you to further restrict the ls output to searching in only the local workspace or searching only on the server. Specifying -workspace provides a faster response to time because the 'ls' command accesses only the local metadata and not the server data. -unlocked shows only objects that are currently unlocked. -locked shows only objects that are currently locked by any user. You can differentiate between objects locked by you and others by noting the fetched state (shown with header "Type"). If you have a lock on the object, the fetched state is "Lock". Specifying both -locked and -unlocked is equivalent to specifying neither option: 'ls' displays both locked and unlocked objects.

-[un]managed (Module-based)

-[un]managed	Determines whether to filter the ls output to show either objects under revision control or objects not under revision control. This option checks the workspace metadata. If the file has been removed on the server, it still displays as managed if the workspace has not been updated. Note: All module members display as managed including added module members that have never been checked in and module members that have been removed and kept in the workspace, if the remove has not been commited to the server. This makes the -unmanaged option irrelevant for modules. When -unmanaged is specified with a module, the server returns an error. To find added or removed members, use ls with the -modified option. -unmanaged shows only objects that are not under revision control. This option is not relevant to modules as mentioned in the previous note. -managed show only objects that are under revision control.

Specifying both -managed and -unmanaged is equivalent to specifying neither option: 'ls' displays both managed and unmanaged objects.

This option only applies to DesignSync objects in workspaces. The option is silently ignored for 'ls' of vault file-based objects.

Note: The url registered command queries the server to determine if the object is managed.

-[un]managed (File-based)

-[un]managed

Determines whether to filter the ls output to show either objects under revision control or objects not under revision control. This option checks the workspace metadata. If the file has been removed on the server, it still displays as managed if the workspace has not been updated.

-unmanaged shows only objects that are not under revision control.

-managed show only objects that are under revision control.

Specifying both -managed and -unmanaged is equivalent to specifying neither option: 'ls' displays both managed and unmanaged objects.

This option only applies to DesignSync file-based objects in workspaces. The option is silently ignored for 'ls' of vault file-based objects.

Note: The url registered command queries the server to determine if the object is managed.

-merged

-merged added|rename|remove all|""

Determines whether to display only objects that have been modified as the result of a merge into into the workspace. You must specify a modifier to -merged. The modifiers behave as follows:

o added - restricts the command output to only those objects added by the merge.

o rename - restricts the command output to only those files that have a different natural path on the merged

branch. These files need to be renamed in order to complete the merge.

o remove - restricts the command output to only those objects that are not present on the merged branch.

o all - includes all the objects specified by the added, removed and renamed modifiers.

o "" - removes the defaults set with the command default system for the -merged option.

-[un]modified (Module-based)

-[un]modified

Determines whether to show only objects that have been modified in the workspace, or only objects that have not been modified in the workspace. These options examine only the workspace for modifications. To compare the workspace against the server to determine whether or not the objects have been modified, use the -[un]changed options.

-unmodified shows only objects that are not modified in the workspace.

-modified show only objects that are modified in the workspace. Unmanaged objects and module members that have been added, removed, or moved, but not checked in are always considered locally modified.

Note: Objects that are "Absent" in the workspace are considered modified.

Specifying both -modified and -unmodified is equivalent to specifying neither option: 'ls' displays both modified and unmodified objects. If -changed is specified with -modified or -unchanged is specified with -unmodified, the -[un]modified option is ignored, because is a subset of the -[un]changed option. If -changed is specified with -unmodified, or -unchanged is specified with -modified, ls returns an error, as these options are mutually exclusive.

This option only applies to workspaces. The option is silently ignored for 'ls' of vault objects.

-[un]modified (File-based)

-[un]modified	Determines whether to show only objects that have been modified in the workspace, or only objects that have not been modified in the workspace. These options examine only the workspace for modifications. To compare the workspace against the server to determine whether or not the objects have been modified, use the -[un]changed options. -unmodified shows only objects that are not modified in the workspace. -modified show only objects that are modified in the workspace. Unmanaged objects are always considered locally modified. Note: Objects that are "Absent" in the workspace are considered modified. Specifying both -modified and -unmodified is equivalent to specifying neither option: 'ls' displays both modified and unmodified objects. If -changed is specified with -modified or -unchanged is specified with -unmodified, the -[un]modified option is ignored, because is a subset of the -[un]changed option. If -changed is specified with -unmodified, or -unchanged is specified with -modified, ls returns an error, as these options are mutually exclusive. This option only applies to workspaces. The option is silently ignored for 'ls' of vault objects.

-modulecontext (Module-based)

-modulecontext <context>	Identifies the module version to operate on. Use the -modulecontext option to restrict the ls to only a particular module if your workspace has overlapping modules so that you can indicate which module you want to run the ls command against. Specify the desired module using the module name (for example, Chip), or a module instance name (for example, Chip%0), or full path to a workspace. Note that you cannot use a -modulecontext option to operate on objects from more than one module; the -modulecontext option takes only one argument, and you can use the -modulecontext option only once on a command line.

-[no]needsmerge

```
-[no]needsmerge      Determines whether to show only objects that
 [-branch <branch>] require a merge or only objects that do not
                    require a merge.  By default, this command
                    compares workspace files against server files in
                    the same branch.  To compare objects against
                    another branch, specify the -branch option.

                    -noneedsmerge shows only objects that do not
                    require a merge.

                    -needsmerge shows only objects that need
                     merging.

                    Note: the -needsmerge option displays objects in
                    which both the server and workspace version of an
                    object indicate changes.  A merge may not
                    actually be possible, depending on the situation.
                    Specifying both -needsmerge and -noneedsmerge is
                    equivalent to specifying neither option: 'ls'
                    lists the objects that need to be merged and
                    those that do not. If -needsmerge is specified
                    with -change or -noneedsmerge is specified with
                    -unchanged, the -[no]needsmerge option is
                    ignored, because is a subset of the -[un]changed
                    option.  If -needsmerge is specified with
                    -changed, or -noneedsmerge is specified with
                    -unchanged, ls returns an error, as these options
                    are mutually exclusive.

                    This option only applies to workspaces.  The option
                    is silently ignored for 'ls' of vault objects.
```

-output

```
 -output <file>      Prints results to named file. The named file is
                     created or overwritten, but not appended to.  To
                     append, use the '-stream' option. The -output and
                     -stream options are mutually exclusive.
```

-path

```
 -path               Include relative paths in object names. By default,
                     only the object name is displayed. With -path,
                     the path of the object relative to the directory
                     specified as an argument during an 'ls
                     -recursive' operation (not necessarily relative to
                     the current directory) is displayed.
                     The -path and -fullpath options are mutually
                     exclusive.
```

-[no]recursive (Module-based)

```
 -[no]recursive      Indicates whether the ls command should operate
                     on the specified argument or all subfolders
                     in the argument's hierarchy.

                     -norecursive operates only on the specified
                     argument. (Default)

                     -recursive operates on all subfolders in the
                     specified argument's hierarchy.

                     If 'ls -recursive' is invoked in a Cadence
                     Cell folder or above, 'ls' does not descend
                     into the Cadence View folders, and so does
                     not list member files, unless the following
                     advanced registry key is set:
                     HKEY_CURRENT_USER\Software\Synchronicity\
                     General\AllowRecursion\Cadence View Folder=dword:1.
                     See DesignSync Data Manager User's Guide:
                     Advanced Registry Settings for details.

                     If ls -recursive is invoked on a module, ls
                     follows the hierarchical references, listing each
                     referenced module separately.
                     Note: ls does not follow hierarchical references
                     to mcache directories, legacy modules, DS vaults,
                     or IPGear deliverables.

                     If the DesignSync site is configured for managed
                     links and 'ls -recursive' is invoked in a
                     directory containing soft links or module caches
                     (Mcaches), 'ls' does not follow the links and
                     instead lists only the objects that are
                     physically present within the directory
                     structure. If the site is configured to treat
                     links as copies of the linked files or
                     directories, 'ls -recursive' does traverse the
                     directory structure.  For more information on
                     managed symbolic links, see the SyncAdmin help.

                     Note: For recursive listings, if multiple objects
                     have the same leaf name, use the -path or
                     -fullpath option to differentiate between these
                     objects.
```

-[no]recursive (Legacy-based)

```
 -[no]recursive      Indicates whether the ls command should operate
                     on the specified argument or all subfolders
                     in the argument's hierarchy.

                     -norecursive operates only on the specified
```

argument. (Default)

-recursive operates on all subfolders in the specified argument's hierarchy.

If 'ls -recursive' is invoked in a Cadence Cell folder or above, 'ls' does not descend into the Cadence View folders, and so does not list member files, unless the following advanced registry key is set: HKEY_CURRENT_USER\Software\Synchronicity\General\AllowRecursion\Cadence View Folder=dword:1. See DesignSync Data Manager User's Guide: Advanced Registry Settings for details.

If 'ls -recursive' is invoked in a directory containing a legacy Hierarchical Configuration Manager (HCM) module configuration, 'ls' follows the hierarchical references and lists the objects referenced in the configuration. In this case, 'ls' does not explicitly indicate that it is listing a legacy module configuration. If 'ls -recursive' is invoked in a subdirectory below the directory containing the legacy module configuration, 'ls' does not follow the hierarchical references; instead 'ls' lists only the objects that are physically present within the directory structure.

If the DesignSync site is configured for managed links and 'ls -recursive' is invoked in a directory containing soft links, 'ls' does not follow the links and instead lists only the objects that are physically present within the directory structure. If the site is configured to treat links as copies of the linked files or directories, 'ls -recursive' does traverse the directory structure. For more information on managed symbolic links, see the SyncAdmin help.

Note: For recursive listings, if multiple objects have the same leaf name, use the -path or -fullpath option to differentiate between these objects.

-[no]recursive (File-based)

-[no]recursive Indicates whether the ls command should operate on the specified argument or all subfolders in the argument's hierarchy.

-norecursive operates only on the specified argument. (Default)

-recursive operates on all subfolders in the specified argument's hierarchy.

If 'ls -recursive' is invoked in a Cadence Cell folder or above, 'ls' does not descend into the Cadence View folders, and so does not list member files, unless the following advanced registry key is set: HKEY_CURRENT_USER\Software\Synchronicity\General\AllowRecursion\Cadence View Folder=dword:1. See DesignSync Data Manager User's Guide: Advanced Registry Settings for details.

If the DesignSync site is configured for managed links and 'ls -recursive' is invoked in a directory containing soft links, 'ls' does not follow the links and instead lists only the objects that are physically present within the directory structure. If the site is configured to treat links as copies of the linked files or directories, 'ls -recursive' does traverse the directory structure. For more information on managed symbolic links, see the SyncAdmin help.

Note: For recursive listings, if multiple objects have the same leaf name, use the -path or -fullpath option to differentiate between these objects.

-report

-report <mode> Specifies what information about each object should be displayed. Available report modes are:

- silent Displays no output (equivalent to '-report !').
- brief Displays just the object name (equivalent to '-report N'). Because no vault information is needed, a brief listing is very fast.
- normal Displays common fields that do not require server vault access (equivalent to '-report MDVLN'). This behavior is the default when -report is not specified.
- verbose Displays most fields (equivalent to '-report OXMSRUNCTA').
- status Displays status fields (equivalent to '-report MSRUN').
- K[K...] Displays the fields corresponding to the data keys, where K is a data key listed in the Report Options table above.
- +K[K...] Displays additional fields corresponding to the data keys specified. This is used to provide addition information

when using a predefined mode such as 'brief' or 'normal'.

- -K[K...] Removes from the display the fields corresponding to the data keys specified. This is used to reduce the amount of information returned when using a predefined mode such as 'brief' or 'normal'.

-[no]selected

-[no]selected	Indicates if the command should use the select list (see the 'select' command) or only the arguments specified on the command line. -noselected indicates that the command should not use the select list. (Default) If -noselected is specified, but there are no arguments selected, the ls command operates on the current directory. -selected indicates that the command should use the select list and any objects specified on the command line. If -selected is not specified, and there are no objects specified on the command line, the ls command operates on the current directory.

-stream

-stream <stream>	Prints results to named Tcl stream. Depending on whether you open the stream using the Tcl 'open' command in write (w) or append (a) mode, you can overwrite or append to an existing file. Note: The -stream option is only applicable in the stcl and stclc Tcl shells, not in the dss and dssc shells. The -output and -stream options are mutually exclusive.

-[non]versionable

-[non]versionable	Determines whether to restrict the report returned to displaying only non-versionable and excluded objects or only objects that are versionable and included. If this option is not specified, all objects, versionable/non-versionable, excluded and included, are displayed. (Default) An object is excluded if it is unmanaged and matches a pattern in an applicable exclude file. Other non-versionable objects include objects that are

members of a versioned collection, which cannot be managed separately.

-[non]versionable displays only the non-versionable and excluded objects

-versionable displays versionable objects only.

For more information on exclude files, see the DesignSync Data Manager User's Guide: Working with Exclude Files.

-workspace/-vault

-workspace | -vault	Determines whether to use only the workspace metadata or query only the vault (server) for the objects being displayed by the ls command. -workspace shows only items that are locked or unlocked in the local workspace. (Default) -vault shows only items present in the local workspace that are locked or unlocked in the vault. Using the -workspace option provides faster results because it does not check the server for objects locked or unlocked outside of the specified workspace, however -vault can provide more complete results.

-writeableunlocked (Module-based)

-writableunlocked	Displays unlocked objects with write access in the workspace. Use -writableunlocked to verify that you have locks on all editable objects, so that you do not accidentally edit an object already locked by another user. Note: If a module branch is locked, all module members in the branch are locked. This option only applies to workspaces. The option is silently ignored for 'ls' of vault objects.

-writeableunlocked (File-based)

-writableunlocked	Displays unlocked objects with write access in the workspace. Use -writableunlocked to verify that you have locks on all editable objects,

```
                    so that you do not accidentally edit an object
                    already locked by another user.

                    This option only applies to workspaces.  The option
                    is silently ignored for 'ls' of vault objects.
```

-xtras (Module-based)

```
 -xtras <list>       List of command line options to pass to the
                     external module change management system. Any
                     options specified with the -xtras option are
                     sent verbatim, with no processing by the
                     populate command, to the Tcl script that defines
                     the external module change management system.
```

--

```
 --                  Indicates that the command should stop looking for
                     command options. Use this option when arguments
                     to the command begin with a hyphen (-).
```

RETURN VALUE

```
 o Empty string if -format value is text.
 o Tcl list if the -format value is list.
 o Empty string if -output or -stream is used with -format.
```

SEE ALSO

```
 ls-foreach, compare, compare-foreach, contents, contents-foreach,
 addhref, edithrefs, select, unselect, vhistory, command defaults,
 hcm showconfs, showhrefs, showmcache, showmods, showstatus, showlocks
```

EXAMPLES

Example Showing the Contents of the Current Folder

```
List the contents of the current folder.  By default, 'ls' reports
data keys MDVLN -- last modified time, workspace status, fetched
version, fetched state (shown as Type), and name.  In this example,
AddBlock is a directory.  SubMod is an unlocked reference, whereas
top.v is a locked reference.

  stcl> ls

  Directory of: file:///home/karen/Projects/Rel40

  Time Stamp       WS Status  Version          Type        Name
  ----------       --------   -------          ----        ----
  04/11/2003 10:13                                        AddBlock
                              Refers to: 1.1   Reference   SubMod
  04/11/2003 09:12 Locally    1.2.1.1          Copy        test.v
                   Modified
                              Refers to: 1.1   Lock        top.v
  04/10/2003 10:16            1.1              Copy        x.v
```

Example Showing the Contents of the Specified Folder

```
List only the specified file and the objects in the ABlk directory
using absolute paths, and add the files to the select list:

  dss> scd /home/Projects
  dss> ls -addselect -fullpath top.v ABlk/*
  Time Stamp       WS Status Version  Type  Name
  ----------       --------- -------  ----  ----
  03/27/2003 11:12           1.1      Lock  file:///home/Projects/ABlk/x.v
  03/27/2003 11:13           1.1      Copy  file:///home/Projects/top.v

  Directory of: file:///home/Projects/ABlk/Add

  Time Stamp       WS Status Version  Type  Name
  ----------       --------- -------  ----  ----
  03/27/2003 11:12           1.1      Copy  file:///home/Projects/ \
  ABlk/Add/Add.v
```

```
dss> select -show
file:///home/Projects/ABlk/x.v
file:///home/Projects/top.v
file:///home/Projects/ABlk/Add/Add.v
```

Example Showing Objects that Need to be Merged

List each object that needs to be merged with its version on the Dev branch:

```
stcl> scd ~/Projects/Rel40
stcl> ls -needsmerge -branch Dev

Directory of: file:///home/karen/Projects/Rel40

Time Stamp       WS Status   Version      Type        Name
----------       ---------   -------      ----        ----
03/27/2003 13:20 Locally     1.3          Copy        test.v
                 Modified
```

Example Showing Objects that do not Need to be Merged

List each object that does not need to be merged with its version on the Dev branch:

```
stcl> ls -noneedsmerge -branch Dev

Directory of: file:///home/karen/Projects/Rel40

Time Stamp       WS Status    Version     Type        Name
----------       ---------    -------     ----        ----
03/27/2003 11:13              1.1         Copy        top.v
```

Example Showing a Recursive Directory Listening

List the changed (not up-to-date) objects in the HTMLHelp folder and all subfolders, and display only the object names (brief format):

```
stcl> ls -recursive -changed -report brief HTMLHelp

Directory of: file:///home/karen/Projects/DesSync/HTMLHelp

Name
----
Customizing_History.htm
DSGetStart_GUI.htm
Editing_and_Organizing_Bookmarks.htm
Get_Tags_Versions.htm
Go_Menu.htm
```

```
  Directory of: file:///home/karen/Projects/DesSync/HTMLHelp/PrintDoc

  Name
  ----
  file.bmp
  unlock.bmp
```

Example Showing the ls Output in List Format

```
Output these changed objects in a list format.  (The list output is
formatted below but doesn't appear that way in the actual list output
unless you list to a file or stream.)

  stcl> ls -recursive -changed -report brief -format list HTMLHelp
  {name HTMLHelp type folder objects
    {
     {name Customizing_History.htm type file}
     {name DSGetStart_GUI.htm type file}
     {name Editing_and_Organizing_Bookmarks.htm type file}
     {name Get_Tags_Versions.htm type file}
     {name Go_Menu.htm type file}
     {name PrintDoc type folder objects
       {
          {name unlock.bmp type file}
          {name file.bmp type file}
       }
     }
   }
  }
```

Example Showing Locked Objects in the Workspace

```
List objects locked in my local workspace.  This command does not
access the SyncServer and does not indicate objects locked by other
users:

  stcl> scd ~/Projects/DesSync/HTMLHelp
  stcl> ls -locked -workspace

  Directory of: file:///home/karen/Projects/DesSync/HTMLHelp

  Time Stamp       WS Status  Version     Type     Name
  ----------       ---------  -------     ----     ----
  03/27/2003 15:06            1.2         Lock     working_folder.htm
```

Example Showing All Locked Objects

```
List objects locked in workspace or by others.  This command accesses
```

```
the SyncServer. The working_folder.htm file is locked in the
workspace, whereas the ocean_arrow_sm.gif file is locked by another
user.

  stcl> ls -locked

  Directory of: file:///home/karen/Projects/DesSync/HTMLHelp

  Time Stamp       WS Status   Version      Type        Name
  ---------        ---------   -------      ----        ----
  03/27/2003 15:05             1.2          Copy        ocean_arrow_sm.gif
  03/27/2003 15:06             1.2          Lock        working_folder.htm
```

Example Showing All Locked Objects with Users

```
List objects locked in workspace and on server. Display using the
'status' report mode which shows the revision control status, the
upcoming version, and the locker of each object:

  stcl> ls -locked -report status

  Directory of: file:///home/karen/Projects/DesSync/HTMLHelp

  Time Stamp       WS Status Server Status Version    Locked By
            Name
  ----------       --------- -------------  -------    ---------
           ----
  03/27/2003 15:05               Up-to-date     1.2 -> 1.3 linda
           arrow_sm.gif
  03/27/2003 15:06               Up-to-date     1.2 -> 1.3 karen*
           work_folder.htm
```

Example Showing Locked Server Objects Using Status Report Mode

```
List only objects locked on the server.  Display using 'status'
report mode which shows the revision control status, the upcoming
version, and the locker of each object.

  This example uses -vault <vaultURL>

dss> ls -locked -vault sync://src:2647/Projects/Help/image
-report status

 Directory of: sync://src:2647/Projects/Help/image

 Time Stamp           WS Status   Server Status  Version   Locked By
 Name
 ----------           ---------   -------------  -------   ---------
 ----
 04/30/2012 14:14                              -           mhopkins
 delete-file.gif;1
 04/30/2012 14:14                              -           mhopkins
```

```
 delete_local_folder.gif;1
 04/30/2012 14:14                                -            mhopkins
 delete_server_folder.gif;1
 04/30/2012 14:14                                -            mhopkins
 delete_vault.gif;1
 04/30/2012 14:14                                -            mhopkins
 delete_version.gif;1
 04/30/2012 14:14                                -            mhopkins
 delete_workspace_mod_dialog.gif;1
```

Example Showing Locked Workspace Objects in Status Report Mode

```
This example uses -vault <workspace name> (in this case "." for
current workspace directory)

dss> ls -locked -vault . -report status

 Directory of:
 file:///c|/Workspaces/Help/image

 Time Stamp             WS Status   Server Status  Version
 Locked By    Name
 ----------             ---------   -------------  -------
 ---------    ----
12/13/2006 13:43                      Up-to-date  1.7 -> 1.8
 mhopkins*    delete-file.gif
12/13/2006 13:43                      Up-to-date  1.5 -> 1.6
 mhopkins*    delete_local_folder.gif
12/13/2006 13:43                      Up-to-date  1.5 -> 1.6
 mhopkins*    delete_server_folder.gif
12/13/2006 13:43                      Up-to-date  1.5 -> 1.6
 mhopkins*    delete_vault.gif
12/13/2006 13:43                      Up-to-date  1.5 -> 1.6
 mhopkins*    delete_version.gif
12/13/2006 13:43                      Up-to-date  1.4 -> 1.5
 mhopkins*    delete_workspace_mod_dialog.gif
```

Example Showing Unmanaged Objects in Current Folder

```
List unmanaged objects in the current folder, displaying only the
name, last-modified time, and size of each file:

  stcl> ls -unmanaged -report MZ

  Directory of: file:///home/karen/Projects/DesSync/HTMLHelp

  Time Stamp                  Size  Name
  ----------                  ----  ----
  04/14/2003 14:03              50  about_ds.htm
```

Example Showing Unlocked Writable Objects in the Workspace

List objects that are writeable but which I have not yet locked:

```
stcl> ls -writableunlocked

Directory of: file:///home/karen/Projects/DesSync/HTMLHelp

Time Stamp       WS Status   Version       Type     Name
----------       ---------   -------       ----     ----
04/14/2003 14:03             Unmanaged              about_ds.htm
03/27/2003 15:06             1.3           Copy     warn_excluded.htm
```

Example Showing Excluding Objects

Exclude objects from a listing:

```
stcl> ls -exclude x.v,top.v

Directory of: file:///home/karen/Projects/Rel40

Time Stamp       WS Status    Version          Type        Name
----------       ---------    -------          ----        ----
04/11/2003 10:13                                          AddBlock
                              Refers to: 1.1   Reference   SubMod
04/14/2003 13:56             Unmanaged                     mult.v
04/15/2003 12:45             Unmanaged                     streamfile
04/11/2003 09:12             1.2.1.1          Copy        test.v

You can also specify the excluded objects using an absolute
URL.  The name in the glob-style expression must match the format
listed, in this case, by using the -fullpath option:

stcl> ls -exclude file:///home/karen/Projects/Rel40/test.v -fullpath

Directory of: file:///home/karen/Projects/Rel40

Time Stamp        ...  Name
----------        ...  ----
04/11/2003 10:13  ...  file:///home/karen/Projects/Rel40/AddBlock
                  ...  file:///home/karen/Projects/Rel40/SubMod
04/14/2003 13:56  ...  file:///home/karen/Projects/Rel40/mult.v
04/15/2003 12:45  ...  file:///home/karen/Projects/Rel40/streamfile
                  ...  file:///home/karen/Projects/Rel40/top.v
04/10/2003 10:16  ...  file:///home/karen/Projects/Rel40/x.v
```

Example Showing a Variety of ls Commands To Display Object Vault

List versions of an object vault. For vault objects, the fetched
state (shown with the "Type" header) is blank:

```
stcl> scd [url vault what_is_dss.htm]
stcl> spwd
```

```
        sync://host:2647/Projects/DS/what_is_dss.htm;
  stcl> ls

        Directory of: sync://host:2647/Projects/DS/what_is_dss.htm;1

        Time Stamp       WS Status  Version  Type  Name
        ----------       ---------  -------  ----  ----
        12/04/2000 16:06            1.1            what_is_dss.htm;1.1
        12/26/2000 16:27            1.2            what_is_dss.htm;1.2
        01/02/2001 17:18            1.3            what_is_dss.htm;1.3
        08/10/2001 11:19            1.4            what_is_dss.htm;1.4
        02/10/2003 13:12            1.5            what_is_dss.htm;1.5

You can instead specify a server-side URL of a vault object to list
its contents:

  stcl> ls sync://host:2647/Projects/DS/what_is_dss.htm\;

Or you can use the 'url vault' command to specify the vault object:

  stcl> ls [url vault what_is_dss.htm]

You can also provide an explicit version number for the vault:

  stcl> ls [url vault what_is_dss.htm]1.3

You can specify a tag for the vault, as well:

  stcl> ls [url vault what_is_dss.htm]Latest

(To determine the existing tags for an object, use '-report T'.)
```

Examples Showing Writing to an Output File or TCL stream

```
Write the list to an output file or Tcl stream.

  This example writes to an output file:

  stcl> ls -output ~/ls_Output
  stcl> cat ~/ls_Output

  Directory of: file:///home/karen/Projects/DesSync/HTMLHelp

  Time Stamp           Version  Type  Name
  ----------           -------  ----  ----
  03/27/2003 15:06     1.12     Copy  About_DesignSync_Log_Files.htm
  03/27/2003 15:06     1.7      Copy  About_Vault_Types.htm
  ...
  ...

The output can be in a list format instead:

  stcl> ls -format list -output ~/ls_Output
  stcl> cat ~/ls_Output
```

```
{
     {
        name HTMLHelp
        type folder
        objects {
            {
                name http_proxy.htm
                type file
                props {
                    fetchedstate Copy
                    mtime {03/27/2003 15:04}
                    version 1.8
                }
            }
 ...
 ...

 This example writes the output to a Tcl stream:

 stcl> set channelid [open streamfile w]
 file8
 stcl> ls -stream $channelid
 stcl> close $channelid
 stcl> cat streamfile

 Directory of: file:///home/karen/Projects/Rel40

 Time Stamp       WS Status  Version         Type       Name
 ----------       ---------  -------         ----       ----
 04/11/2003 10:13                                       AddBlock
                             Refers to: 1.1  Reference  SubMod
 04/14/2003 13:56            Unmanaged                  mult.v
 ...
 ...
```

Example Showing Locked References

```
List locked references.

  You might need to regenerate managed objects, in which
  case you can check them out as 'locked references'
  rather than actual locked copies of the objects.  The
  example below creates a locked reference, top.v. The
  top.v fetched state displays as 'Lock' and the
  Version displays as 'Refers to:' followed by the
  version:

  stcl> co -reference -lock -nocomment top.v

  Beginning Check out operation...

  Checking out: top.v : Locked Reference made to 1.1.

  Checkout operation finished.
```

```
{Objects succeeded (1)} {}

stcl> ls

Directory of: file:///home/karen/Projects/Rel40

Time Stamp       WS Status  Version         Type    Name
----------       ---------  -------         ----    ----
03/27/2003 11:12                                    AddBlock
04/03/2003 10:46            1.3             Copy    test.v
                            Refers to: 1.1  Lock    top.v
```

Example Showing Collection List

```
For each member of a collection, list the object type and the
owner (the collection to which the member belongs). This example
  uses a Custom Type Package (CTP) collection.

  stcl> ls -report OX

  Directory of: file:///home/karen/sf242data/jul16/coltest

  Object Type         Name
  -----------         ----
  File                README
  a Test Member       a.html
  Owner: /home/karen/sf242data/jul16/coltest/a.sgc.tst

  File                a.prop
  a Test collection   a.sgc.tst
  a Test Member       a.txt
  Owner: /home/karen/sf242data/jul16/coltest/a.sgc.tst

  File                b.prop
  File                b.txt
  File                c.html
  d Test Member       d.html
  Owner: /home/karen/sf242data/jul16/coltest/d.sgc.tst

  File                d.prop
  d Test collection   d.sgc.tst
  d Test Member       d.txt
  Owner: /home/karen/sf242data/jul16/coltest/d.sgc.tst

  f Test Member       f.html
  Owner: /home/karen/sf242data/jul16/coltest/f.sgc.tst

  File                f.notamember
  File                f.prop
  f Test collection   f.sgc.tst
  f Test Member       f.txt
  Owner: /home/karen/sf242data/jul16/coltest/f.sgc.tst

  g Test Member       g.html
```

```
  Owner: /home/karen/sf242data/jul16/coltest/g.sgc.tst

  File               g.prop
  g Test collection  g.sgc.tst
  g Test Member      g.txt
  Owner: /home/karen/sf242data/jul16/coltest/g.sgc.tst

  File               partnerFile
```

Example Showing Module Structural Changes (Module-based)

```
Lists the objects in a module containing structural changes
consisting of an added file, documentstyles.css, a removed file,
c.txt, and a removed file, b.txt that was retaining in the workspace
with the -keep option, and a moved file, chipintro.doc.

stcl> ls

 Directory of: file:///e|/workspaces/X5Mods/chip/doc

 Time Stamp        WS Status  Version         Type   Name
 ----------        ---------  -------         ----   ----
 11/19/2008 09:08  Removed    1.3                    b.txt
                   Removed    Refers to: 1.2         c.txt
 11/19/2008 09:08  Moved      1.2              Copy  chipintro.doc
  Original path: \doc\chip.doc
 11/19/2008 09:08             1.1              Copy  commands.html
 11/20/2008 16:25  Added                             documentstyles.css
 11/19/2008 09:08                                    images
 11/19/2008 09:08             1.1              Copy  index.html
 11/19/2008 09:08             1.1              Copy  interface.html
 11/19/2008 09:08             1.1              Copy  manual.pdf
```

Example Showing the Contents of a Legacy Module Configuration (Legacy-based)

```
List legacy module configuration contents.

A workspace directory, /home/ian/ws/modtop, is populated with HCM
module configuration ModTop@RelA.  There is one submodule, SubMod1,
with relative path submods/submod1. Notice that the submodule of the
ModTop@RelA configuration are listed with their relative pathnames in
the modtop listing (submods/submod1), and their contents are shown
below:

  stcl> ls -recursive

  Directory of: /home/ian/ws/modtop

  Time Stamp      WS Status Version    Type    Name
  ----------      --------- -------    ----    ----
  01/29/03 09:01            1.2        Copy    file1.txt
  01/29/03 09:01                               submods
```

```
01/29/03 08:30                                  submods/submod1
01/29/03 10:25            Unmanaged             tmp.txt

Directory of: /home/ian/ws/submods

Time Stamp      WS Status Version    Type    Name
----------      --------- -------    ----    ----
01/29/03 08:20            1.4        Mirror  file4.txt
01/29/03 05:50                               psref
01/29/03 08:30                               submod1

Directory of: /home/ian/ws/submods/psref

Time Stamp      WS Status Version    Type    Name
----------      --------- -------    ----    ----
01/29/03 05:50            1.3        Copy    file5.txt

Directory of: /home/ian/ws/submods/submod1

Time Stamp      WS Status Version    Type    Name
----------      --------- -------    ----    ----
01/29/03 08:40            1.3.1.2    Copy    file3.txt

The equivalent return structure if the -format list option were given is:

{
{
    name /home/ian/ws/modtop
    type folder
    objects {
    {
         name submods
         type folder
         props {
            mtime {01/29/03 09:01}
         }
         objects {
            {
               name file4.txt
               type file
               props {
                  mtime {01/29/03 08:20}
                  version 1.4
                  fetchedstate Mirror
               }
            }
            {
               name psref
               type folder
               props {
                  mtime {01/29/03 05:50}
               }
               objects {
                  {
                     name file5.txt
                     type file
                     props {
```

```
                        mtime {01/29/03 05:50}
                        version 1.3
                        fetchedstate Copy
                     }
                  }
               }
            }
         }
         {
             name submod1
             type folder
             props {
                mtime {01/29/03 08:30}
             }
             objects {
                {
                   name file3.txt
                   type file
                   props {
                      mtime {01/29/03 08:40}
                      version 1.3.1.2
                      fetchedstate Copy
                   }
                }
             }
         }
      }
      {
         name tmp.txt
         type file
         props {
            mtime {01/29/03 10:25}
            version Unmanaged
         }
      }
   }
}
}
```

ls-foreach

ls-foreach Command

NAME

```
ls-foreach          - Function to process the results of an ls
                      command
```

DESCRIPTION

```
This routine loops over the items in an "ls" results list, and
processes each item in turn.
```

```
Note: The object type, identified by reporting on the O key in the ls
command, is identified by the otype property name.  This
distinguishes it from the type property reported automatically.  The
type property reports whether the object is a folder, file, or
module.
```

SYNOPSIS

```
ls-foreach var result_list tcl_script [-nofolder] [-path]
```

ARGUMENTS

- Loop Variable
- List of Objects to be Processed
- TCL script

Loop Variable

```
var             This is the loop variable. It is treated as a Tcl
                array, and on each loop around contains the set of
                properties for the next object in the result_list.
                In addition to the properties in the "props" value
                for each object, the array will contain a "name"
                property and a "type" property, which are the name
                and type properties for the object.
```

List of Objects to be Processed

```
result_list     This is the list of objects to be processed. It must
                be the result value of a call to the "ls" command
                with the "-format list" option.
```

TCL script

```
tcl_script      This is the piece of Tcl code that is executed on
                each loop.
```

OPTIONS

- -nofolder
- -path

-nofolder

```
-nofolder        If specified, then the tcl_script will not be called
                    for Folder type objects in the result_list.
```

-path

```
-path            The "name" property on each loop is usually just
                 the "name" property for the object. However, if this
                 option is specified, and a recursive "ls" was
                 performed, then the "name" property is the relative
                 path to each object. Normally, you would run "ls"
                 with the -path or -fullpath option, in which case the
                 "name" property contains an appropriate relative or
                 full path. If you did not do that, then passing the
                 "-path" option to ls-foreach will mean that the
                 "name" property contains the relative path for each
                 item, thus allowing you to differentiate between
                 items with the same name in different folders.

The set of properties available for each object is dependant on the
"-report" option passed to the "ls" command.
```

SEE ALSO

```
ls
```

EXAMPLE

```
This shows a sample script that creates an array to run ls-foreach
against and extracts the object name and otype.  The output shown
after the query are the results of running the query against a folder
containing Cadence data collections.

set result_list [ls -rec -format list -report normal+O]

ls-foreach obj $result_list {
 if {[info exists obj(otype)]} {
 puts "OBJ: $obj(name), OTYPE: $obj(otype)"
}
}

 OBJ: cdsinfo.tag, OTYPE: Cadence Info File
 OBJ: rec, OTYPE: Cadence Cell
 OBJ: schematic.sync.cds, OTYPE: Cadence View
 OBJ: schematic, OTYPE: Cadence View Folder
 OBJ: mux2, OTYPE: Cadence Cell
 OBJ: schematic.sync.cds, OTYPE: Cadence View
 OBJ: schematic, OTYPE: Cadence View Folder
 OBJ: celdom, OTYPE: Cadence Cell
 OBJ: symbol.sync.cds, OTYPE: Cadence View
```

```
OBJ: symbol, OTYPE: Cadence View Folder
OBJ: custinv, OTYPE: Cadence Cell
OBJ: symbol.sync.cds, OTYPE: Cadence View
OBJ: symbol, OTYPE: Cadence View Folder
OBJ: .oalib, OTYPE: File
OBJ: risk.TopCat, OTYPE: Cadence Lib Category
...
OBJ: schematic_v1#2e1, OTYPE: Cadence NonView Folder
```

syncinfo

syncinfo Command

NAME

```
syncinfo            - Returns Synchronicity environment information
```

DESCRIPTION

```
This command returns information about the Synchronicity
software environment, such as version number, location of
registry files, and default editor and HTML browser. The command
can be run from the client to return client information, or from
the server to return server information.

By default (with no arguments specified), all available information
is returned. You can request specific information by specifying
one or more command arguments.

If a given value has not been set or is not available, then
'syncinfo' returns an empty string. For example, if you ask for
portRegistryFile from the client, the return value is empty because
portRegistryFile is only available from the server.
```

SYNOPSIS

```
syncinfo [<arg> [<arg>...]]
```

ARGUMENTS

- General Information
- isServer
- syncDir
- version
- Registry Information
- clientRegistryFiles

General Information

isServer

```
isServer            Returns a Tcl boolean value (0 or 1) indicating
                    whether the software executing the syncinfo
                    command is acting as a server (1) or client (0).
```

syncDir

```
syncDir             Returns the root directory of the Synchronicity
                    software installation. On UNIX, this value
                    corresponds to the SYNC_DIR environment
                    variable (on Windows, SYNC_DIR is not required).
```

version

```
version             Returns the version of the Synchronicity software
                    as a string.
```

Registry Information

clientRegistryFiles

```
clientRegistryFiles    Returns a comma-separated list of registry
                       files used by the Synchronicity clients
                       (DesSync, stcl, dss, stclc, dssc).
```

enterpriseRegistryFile

```
enterpriseRegistryFile Returns the enterprise-wide registry file.
```

portRegistryFile

```
portRegistryFile       Returns the port-specific registry file.
```

projectRegistryFile

```
projectRegistryFile    Returns the project-specific registry file.
```

serverRegistryFiles

```
serverRegistryFiles    Returns a comma-separated list of registry
                       files used by a Synchronicity server.
```

siteRegistryFile

```
siteRegistryFile       Returns the site-wide registry file.
```

syncRegistryFile

```
syncRegistryFile        Returns the Synchronicity-supplied standard
                        registry file.
```

userRegistryFile

```
userRegistryFile        Returns the user-specific registry file.
```

usingSyncRegistry

```
usingSyncRegistry       Returns a Tcl boolean value (0 or 1)
                        indicating whether the Synchronicity
                        software is using the text-based registry (1)
                        or the native Windows registry (0).
```

Customization Information

customDir

```
customDir           Returns the root directory of the 'custom' branch
                    of the Synchronicity installation hierarchy,
                    which contains all site- and server-specific
                    customization files. The default value,
                    <SYNC_DIR>/custom, can be overridden by the
                    SYNC_CUSTOM_DIR environment variable.
```

customSiteDir

```
customSiteDir       Returns the directory that contains site-specific
                    customization files. The default value,
                    <SYNC_CUSTOM_DIR>/site (which defaults to
                    <SYNC_DIR>/custom/site), can be overridden by
                    the SYNC_SITE_CUSTOM environment variable.
```

customEntDir

```
customEntDir        Returns the directory that contains enterprise-specific
                    configuration files. The default value,
                    <SYNC_ENT_CUSTOM> (which defaults
                    to <SYNC_CUSTOM_DIR>/enterprise),
                    can be overridden by the SYNC_ENT_CUSTOM
                    environment variable.
```

siteConfigDir

siteConfigDir	Returns the directory that contains site-specific configuration files. The default value, <SYNC_SITE_CUSTOM>/config (which defaults to <SYNC_CUSTOM_DIR>/site/config, which defaults to <SYNC_DIR>/custom/site/config), can be overridden by the SYNC_SITE_CNFG_DIR environment variable.

usrConfigDir

userConfigDir	Returns the directory that contains user configuration files. The default value, <HOME>/.synchronicity, can be overridden by the SYNC_USER_CFGDIR environment variable.

userConfigFile

userConfigFile	Returns the user configuration file. The default value, <HOME>/.synchronicity/user.cfg, can be overridden by the SYNC_USER_CONFIG environment variable.

Client Information

connectTimeout

connectTimeout	Returns the number of seconds the client will wait per communication attempt with the server.

commAttempts

commAttempts	Returns the number of times client/server communication is attempted before failing. Using multiple attempts protects against transient network problems. 'Connect Failure' failures do not trigger multiple connection attempts, because transient network problems rarely cause this error. Note: When the number of communication attempts is the default value of 3, 'syncinfo commAttempts' returns no value instead of returning 3.

defaultCache

defaultCache	Returns the default cache directory for the client as specified during installation or using SyncAdmin.

fileEditor

fileEditor	Returns the default file editor as specified during installation or using SyncAdmin.

htmlBrowser

htmlBrowser	(UNIX only) Returns the default HTML browser as specified during installation or using SyncAdmin.

proxyNamePort

proxyNamePort	Returns the <name>:<port> of a proxy, if one is defined in a client registry file or using the ProxyNamePort environment variable.

somTimeout

somTimeout	Returns the number of milliseconds after an unsuccessful server connection attempt during which the client does not try to connect again. This timeout protects against an operation on many objects (such as 'ls' on a large directory) taking an excessively long time to complete when there is a connection failure (such as when the server is down). Instead of waiting the connectTimeout period for each object, the operation fails for all objects after the first connection failure.

Server Information

berkdbIsShmEnabled

berkdbIsShmEnabled	For Synchronicity internal use only.

berkdbShmKey

berkdbShmKey	For Synchronicity internal use only.

isTestMode

isTestMode	For Synchronicity internal use only. Returns a Tcl boolean value (0 or 1) indicating whether the software executing the syncinfo command is running in test mode (1) or not (0). This feature is useful for regression testing of servers.

serverMetadataDir

serverMetadataDir	Returns the directory that contains the server metadata (such as relational database) files.

serverDataDir

serverDataDir	Returns the directory that contains vault (repository) data that is stored by a server.

serverMachine

serverMachine	Returns the name of the server as returned by gethostname(). This value is returned only when 'syncinfo' is run from a server-side script.

serverName

serverName	Returns the name of the server as it was specified in the URL used to contact the server. This value is returned only when 'syncinfo' is run from a server-side script.

serverPort

serverPort	Returns the port number used by the server to respond to the syncinfo request. This value is returned only when 'syncinfo' is run from a server-side script.

User Information

home

```
home                Returns the home directory of the user
                    running syncinfo (HOME on UNIX, or as
                    defined in your user profile on Windows platforms).
```

userName

```
userName            Returns the account name of the user
                    running syncinfo.
```

RETURN VALUE

```
In dss/dssc mode, you cannot operate on return values, so the return
value is irrelevant.

In stcl/stclc mode:
  - If no argument is specified, the return value is a
    name/value list (Tcl 'array get' format) containing
    all available information.
  - If a single argument is specified, the return value is
    the requested value (not a list).
  - If more than one argument is specified, the return value
    is a name/value list containing the requested information.
  - If any argument is not known, an exception is thrown.
```

SEE ALSO

```
server-side
```

EXAMPLES

- Example Showing the SyncInfo Version on Client Startup
- Example of Extracting SyncInfo Information to an Array
- Example Showing Extracting the Information from an Array
- Example of extracting Name/Value Pairs for Specific Arguments

Example Showing the SyncInfo Version on Client Startup

```
When you start any Synchronicity client, 'syncinfo version'
executes, which displays (and writes to your log file
if logging is enabled) the Synchronicity version. In this
example, the software is version 3.0.
  % stclc
  Logging to c:\goss\dss_01192000_092559.log
```

```
V3.0

stcl>
```

Example of Extracting SyncInfo Information to an Array

```
The following stcl script fragment shows how to get all known
information as a Tcl array variable. The 'version' string is
then printed.
  array set info [syncinfo]
  puts "Version: $info(version)"
```

Example Showing Extracting the Information from an Array

```
This example uses the single-argument form of syncinfo to print the
same version information provided by the previous example:

  puts "Version: [syncinfo version]"
```

Example of extracting Name/Value Pairs for Specific Arguments

```
The following example uses command arguments to return a list
of the 'syncDir' and 'userName' values. This example
also shows how to enumerate the name/value list returned by
syncinfo without storing it in an array variable.
  foreach {name value} [syncinfo syncDir userName] {
      puts "$name: $value"
  }
```

version

hcm version Command

NAME

```
hcm version         - Displays the DesignSync installation version
```

DESCRIPTION

```
This command displays the version of the DesignSync product
installation.
```

SYNOPSIS

```
hcm version
```

OPTIONS

RETURN VALUE

```
This command does not return Tcl values.
```

SEE ALSO

```
syncinfo
'
```

EXAMPLES

```
The following example displays the DesignSync version information.

dss> hcm version
 V6R2012
```

vhistory

vhistory Command

NAME

```
vhistory            - Displays an object's version history
```

DESCRIPTION

- Reporting on Modules (Module-based)
- Report options (Module-based)
- Understanding the output (Module-based)
- Report options (File-based)
- Understanding the output (File-based)

```
The vhistory command reports version history for managed objects. If
the command is run from a workspace, local status is also reported.

This command supports the command defaults system.
```

Reporting on Modules (Module-based)

Running vhistory on a module reports the history of that module.

Note: To list the module versions that contain a module member, use the whereused member command.

Attempting to run vhistory on a module folder will report an error, instructing you to run the command on a module instead.

Module members are managed within the context of their parent module. When you run vhistory on a module member object, it shows the only the module versions in which the specified module member has been directly affected by actions performed on the module, such as content change, tag, rename, remove, etc. This is noted in the Report Options table below, for the Module Manifest.

The vhistory command does not recurse through module hierarchy. If a module is being reported on, and the -recursive option was specified, the vhistory command will output a warning.

When vhistory is run with the -module option, and two or more module members are specified, a single vhistory report is produced that contains all the module version in which any of the module members specified have been modified.

Note: If you have specified two or more module members and the lastversions options, you may only see one of the module members reported if the other does not have any versions in the specified timeline.

Report options (Module-based)

The -report option lets you specify what information vhistory reports. You can specify:

- One of the predefined modes (silent, brief, normal, verbose).
- One or more data keys, to define exactly the information you want.
- A combination of data keys to add to, or remove from, a predefined report.

The predefined report modes, and how to modify them for a single vhistory invocation, are described in the "-report" option description.

The following table lists the -report data keys, including the corresponding property names used in "-format list" output. Note that all data keys must be uppercase.

Text Label	Data Key	Property Name	Description
Object:	N	name	The workspace path to the object, or to the vault URL.
===== -----	H	N/A	Show horizontal separators between items and versions.
Vault URL	S	url	Show the vault URL (server address) associated with a workspace object.
Current version	W	version	Show the version currently in the workspace.
Current	L	state	Show the fetched state in the workspace. This is not reported for module data.
	B	N/A	Show entries for the branch objects. Note:
	R	N/A	Show entries for the version objects.
	I	N/A	Do not show (ignore) entries that have no tags.
Branch tags/Version tags	T	tags	Show the branch and version tags. Immutable tags are shown with "(immutable)" appended only if Y is specified as well..
Tag comments	Y	tag_properties	Show all properties associated with the version and branch tags including the tag dates, tag comments, and an "(immutable)" notation if the tag is immutable. In "-format list" output, the property value is a list of five values. Each set of values consists of a tag name, 0 or 1 indicating whether the tag is immutable, the tag comments, the tag date, and the user who created the tag.
Version	V	version, bud	Show the version numbers for versions, and the branch number for branches. Also, for branches, the property "bud" will be included. A branch is a "bud" branch if it does not yet have any versions. A value of "1" indicates the branch is a bud branch, else "0".
Date	D	date	Show the creation date for a version.
Derived from	F	derived_from	Show the numerical parent version. This maintains the continuity between

			versions for merge and rollback operations. Note: If a merge, skip,rollback or overlay operation occurs to create this version, the referenced version is shown as "Merged from" version.
Author	A	author	Show the author of a version.
Size	K	size	Show the size of the object version in KB. Note: Collections and module versions, both of which contain more than one object, display with a size of zero.
Merged from	E	merged_from	Show the version used to create the current version when the current version was created as the result of a rollback, merge, skip, or overlay operation requiring an alternate parent version.
Comment	C	comment	Show the checkin comments for a version, and any checkout comments. For DesignSync objects, checkout comments are only visible from the workspace in which the checkout occurred. For module objects, the branch lock comment is visible to all users.
Locked by	U	locker, upcoming	Show the lock owner of a locked branch. The text and list formats both show the latest version and leaves the upcoming version blank.
Version graph	G	N/A	Show a graphical representation of the version history, as a text graph.
Reverse order	Z	N/A	Show the versions/branches in reverse numeric order.
Module	Q	manifest	Show the manifest of Manifest changes in each version. For a module member, show only the changes to that member. Note: When a module rollback has been performed, the changes between versions are the changes that were "rolled back." In "-format list" output, the property value is a list of property lists, with one entry for each change recorded in the module version.
Tagged	M	N/A	Include Module version that have tags,

Module Version			even if a module member being queried has not been changed in that module version.
N/A	P	deleted	Includes deleted module versions with the information that the module has been deleted. Note: Appears in the text layout as "This version has been deleted."
N/A		objects	In "-format list" output, the property value is a list of the branch and version items reported for that object. Each entry in the objects value is itself a property list.
N/A		type	In "-format list" output, the property value is either "branch" (for branch entries) or "version" (for version entries). The value is used in the "objects" property value lists.
N/A	+	N/A	Add codes to a predefined report.
N/A	-	N/A	Remove codes from a predefined report.

Understanding the output (Module-based)

The vhistory output is divided into sections. The first section provides the information about the selected module. The second section contains branch information for the currently selected branch, followed by the version information of all versions on that branch. If you have requested information about more than one branch, the branch section, ordered by branch number, is displayed, followed by the versions on that branch; followed by the next branch sequentially, etc. until all specified branches and versions have been enumerated. The sequence is based on depth of the branch and version numbers, for example the branch number 1.2.4.1 appears after branch 1.2.3, but before 1.3. The final section is the history graph.

Notes: The sections and fields that appear in your report depend on the report formats you select. For more information on any of the displayed fields, see the Report options section.

Object information can include the following fields:

o Object - Workspace path to the object..
o Vault URL - Vault URL associated with the object.
o Current Version - Version number of the workspace version.

Branch information includes the following fields:

Note: You must include report option B to get information on

```
branches.  Additional options determine what branch information you
display.

o Branch - Branch number.
o Branch tags - Branch tag names.
o Branch tag properties - Immediately following the appropriate
  branch tag, the following information is also displayed:
      - "immutable" when the tag is immutable.
      - tag application date
      - username of the operator who applied the tag
      - tag comment, if applicable, on the following line.
o Locked by - username of the branch locker.
o Comment - Comment applied to the branch during creation.  For the
  Trunk branch, this is the comment entered when the module was
  created.

Version information includes the following fields:

Note: You must include the report option R to get information on
versions. Additional options determine what version information you
display.

o Version - Version number.
o Version tags - Version tag names.
o Version tag properties - Immediately following the appropriate
  version tag, the following information is also displayed:
      - "immutable" when the tag is immutable.
      - tag application date
      - username of the operator who applied the tag
      - tag comment, if applicable, on the following line.
   Note: If the tag was applied with a checkin, the tag
   properties information is identical to Date, Author, Comment
   fields.
o Derived From - numeric parent version.
o Merged From - version used to create the current version.
o Date - version creation date.
o Author - version author.
o Comment - version comment.
o Module Manifest - list of files and hierarchical references changed
  in the version.

History graph information includes the following:
A graphical representation of the object's history.
```

Report options (File-based)

```
The -report option lets you specify what information vhistory
reports. You can specify:

o One of the predefined modes (silent, brief, normal, verbose).

o One or more data keys, to define exactly the information you want.

o A combination of data keys to add to, or remove from, a
```

predefined report.

The predefined report modes, and how to modify them for a single vhistory invocation, are described in the "-report" option description.

The following table lists the -report data keys, including the corresponding property names used in "-format list" output. Note that all data keys must be uppercase.

Text Label	Data Key	Property Name	Description
Object:	N	name	The workspace path to the object, or to the vault URL.
===== -----	H	N/A	Show horizontal separators between items and versions.
Vault URL	S	url	Show the vault URL (server address) associated with a workspace object.
Current version	W	version	Show the version currently in the workspace.
Current	L	state	Show the fetched state in the workspace. This is not reported for module data.
	B	N/A	Show entries for the branch objects. Note: When used with the B option, on a V6R2010 or higher SyncServer, also reports the username and timestamp for retired branches.
	R	N/A	Show entries for the version objects.
	I	N/A	Do not show (ignore) entries that have no tags.
Branch tags/Version tags	T	tags	Show the branch and version tags. Immutable tags are shown with "(immutable)" appended only if Y is specified as well..
Version	V	version, bud	Show the version numbers for versions, and the branch number for branches. Also, for branches, the property "bud" will be included. A branch is a "bud" branch if it does not yet have any versions. A value of "1" indicates the branch is a bud branch, else "0".
Date	D	date	Show the creation date for a version.
Derived from	F	derived_from	Show the numerical parent version. This maintains the continuity between

			versions for merge operations. Note: If a merge, or overlay operation occurs to create this version, the referenced version is shown as "Merged from" version.
Author	A	author	Show the author of a version.
Size	K	size	Show the size of the object version in KB. Note: Collections which contain more than one object, display with a size of zero.
Merged from	E	merged_from	Show the version used to create the current version when the current version was created as the result of a merge, skip, or overlay operation requiring an alternate parent version.
Comment	C	comment	Show the checkin comments for a version, and any checkout comments. For DesignSync objects, checkout comments are only visible from the workspace in which the checkout occurred.
This branch is retired	X	retired	Show whether a branch is retired. A "retired" value of "1" indicates the branch is retired, else "0". Note: When used with the B option, on a V6R2010 or higher SyncServer, also reports the username and timestamp for retired branches.
Locked by	U	locker, upcoming	Show the lock owner of a locked branch. For DesignSync objects, also show the "version -> upcoming version" information.
Version graph	G	N/A	Show a graphical representation of the version history, as a text graph.
Reverse order	Z	N/A	Show the versions/branches in reverse numeric order.
N/A		objects	In "-format list" output, the property value is a list of the branch and version items reported for that object. Each entry in the objects value is itself a property list.
N/A		retired_properties	In "-format list" output, the property value is an array including date and user properties containing the date and time the retire was performed and the

			username of the person who performed the retire.
N/A		type	In "-format list" output, the property value is either "branch" (for branch entries) or "version" (for version entries). The value is used in the "objects" property value lists.
N/A	+	N/A	Add codes to a predefined report.
N/A	-	N/A	Remove codes from a predefined report.

Understanding the output (File-based)

The vhistory output is divided into sections. The first section provides the information about the selected object or module. The second section contains branch information for the currently selected branch, followed by the version information of all versions on that branch. If you have requested information about more than one branch, the branch section, ordered by branch number, is displayed, followed by the versions on that branch; followed by the next branch sequentially, etc. until all specified branches and versions have been enumerated. The sequence is based on depth of the branch and version numbers, for example the branch number 1.2.4.1 appears after branch 1.2.3, but before 1.3. The final section is the history graph.

Notes: The sections and fields that appear in your report depend on the report formats you select. For more information on any of the displayed fields, see the Report options section.

Object information can include the following fields:

- Object - Workspace path to the object..
- Vault URL - Vault URL associated with the object.
- Current Version - Version number of the workspace version.
- Current State - Fetched state in the workspace.

Branch information includes the following fields:

Note: You must include report option B to get information on branches. Additional options determine what branch information you display.

- Branch - Branch number.
- Branch tags - Branch tag names.
- Locked by - username of the branch locker.
- Comment - Comment applied to the branch during creation. For the Trunk branch, this is the comment entered when the module or DesignSync object was created.
- This Branch is Retired. - Object branch has been retired, (Non-module data only)
- Retired by - Username, date, and time associated with the retire.

```
Version information includes the following fields:

Note: You must include the report option R to get information on
versions. Additional options determine what version information you
display.

o Version - Version number.
o Version tags - Version tag names.
o Derived From - numeric parent version.
o Merged From - version used to create the current version.
o Date - version creation date.
o Author - version author.
o Comment - version comment.

History graph information includes the following:
A graphical representation of the object's history.
```

SYNOPSIS

```
vhistory [-branch <branchname> -descendants <n> |
          -lastversions <n> -lastbranches <n> | -all]
         [-exclude <string>] [-format list | text] [-maxtags <n>]
         [-modulecontext <context>]
         [-output <filename> | -stream <port>] [-report <mode>]
         [-[no]recursive] [-[no]selected] [-xtras <list>] [--]
         <argument> [<argument>...]
```

ARGUMENTS

- Module Member (Module-based)
- Workspace Module (Module-based)
- Server Module (Module-based)
- DesignSync Object (File-based)
- Workspace Folder (File-based)
- Server Folder (File-based)
- legacy_note (Legacy-based)

```
Specify one or more of the following arguments:
```

Module Member (Module-based)

```
<module member>     Specifies the module member.
```

Workspace Module (Module-based)

```
<workspace module>     Specifies the workspace module.  You may
                       specify a module instance name or a full
                       module address. It is compared against the
                       corresponding server module.
```

Server Module (Module-based)

```
<server module>     Server modules can be selected using the URL of
                    the module.
                    sync[s]://<host>[:<port>]/<vaultPath> where
                    <host> is the SyncServer on which the module
                    resides, <port> is the SyncServer port number,
                    and <vaultPath> identifies the module to select.
```

DesignSync Object (File-based)

```
<DesignSync object> Specifies the DesignSync object.
```

Workspace Folder (File-based)

```
<Workspace folder>  Specifies the history of the contents of the
                    specified folder, and, when used with the
                    recursive option, all subfolders.
```

Server Folder (File-based)

```
<server folder>     Specifies the history of the contents of the
                    specified folder on the server, and when used
                    with the -recursive option, all
                    subfolders. Specify the object with the sync URL
                    in the format:
                    sync://<host>:<port>/<path>/<folder>
```

```
If no arguments are given, and the -selected option is not specified,
then the vhistory command will operate on the current directory. This
is equivalent to specifying a single argument of "."
```

```
Note: A legacy module is regarded as a DesignSync folder.
```

OPTIONS

- -all
- -branch
- -descendants

-all

```
-all            Report branch "1" and all descendants, thereby
                reporting the entire history of an object.

                The "-all" option is mutually exclusive with the
                "-descendants" option, the "-lastversions" option, the
                "-lastbranches" option, and with the "-branch" option.
```

-branch

```
-branch <branchname>
                Start the report at the specified branch name. The
                <branchname> may be a branch tag or a branch numeric.

                By default, the current branch for workspace objects
                is the starting branch. For vault objects, branch 1 is
                the default starting branch.

                To override a default value that was saved with the
                command default system, specify a value of "". That
                will use the aforementioned default behavior.
```

-descendants

```
-descendants <n>
                The number of levels of descendant branches to report,
                from the starting branch. By default, the report is
                limited to the starting branch (an <n> value of 0).

                You may specify any positive number as the <n> value.

                For example, if branch 1.2.1 is being reported on,
                and the descendants value is 1, then branch 1.2.1.3.1
```

```
                will be reported, but branch 1.2.1.3.1.4.1 will not be.

                Specifying a value of "all" will report all levels.

                The -descendants option is mutually exclusive with the
                -lastversions option and with the -lastbranches option.
```

-exclude

```
 -exclude <string>
                Specifies a glob-style expression to exclude matching
                object names from the report. The string you specify
                must match the name of the object as it would have
                appeared in the listing.

                By default, the vhistory command does not exclude the
                objects in the global exclude lists (set using
                Tools->Options->General->Exclude Lists or using
                SyncAdmin's General->Exclude Lists). To exclude these
                objects from a vhistory report, apply the -exclude
                option with a null string:
                        dss> vhistory -exclude ""
                The objects in the global exclude lists are appended
                to the vhistory exclude list if you exclude other
                values:
                        dss> vhistory -exclude "README.txt"
```

-format

```
 -format
                Specifies whether the "vhistory" command generates a
                formatted report, or returns a Tcl property list.

                list   Returns a list, with each result entry
                       containing the properties reported for each
                       object, and an "objects" property. The objects
                       property contains a sublist of property lists,
                       with one entry for each branch and version
                       object that is reported for the parent object.

                       For example, consider the following command:

                       stcl> vhistory -report LNRVT file1.txt \
                             file2.txt -lastversions 2 -format list

                       The above command requests a report of the
                       last two versions on the current branch of the
                       two specified objects. The report will contain
                       the object name, the state of the objects in
                       the workspace, and the versions of the object.
                       For each version, the version number and any
                       tags are reported.
```

```
                        The result might be:

                        {
                          name file:///home/tbarbg10/Test/file1.txt
                          state Copy
                          objects {
                            {type version version 1.4 tags {t1 t2}}
                            {type version version 1.5 tags {t3 Latest}}
                          }
                        }

                        {
                          name file:///home/tbarbg10/Test/file2.txt
                          state Lock
                          objects {
                            {type version version 1.3.1.5 tags {}}
                            {type version version 1.3.1.6 tags Latest}
                          }
                        }

                        As shown above, the result is a list containing
                        one entry for each object for which the history
                        was requested.

                        To process the results, use the
                        vhistory-foreach and vhistory-foreach-obj
                        functions.

                        If the history was requested for a single
                        object, you must start processing the result
                        list by taking the "head" of the list, with a
                        call such as "[index $result 0]".

                        The property lists will always contain a
                        property even if the value is "", for easier
                        processing of the results.

                        For a list of properties, see the Report Options
                        table above.

                    text   Display a textual result. (Default)
```

-lastbranches

```
 -lastbranches <n>
                    How many branches back to report. By default, only
                    versions on the specified branch are reported (an
                    <n> value of 0).

                    You may specify any positive number as the <n> value.
                    <n> parent branches back will be reported on. This
                    option is used to show more of an object's history.
```

For example, let's say the branch to be reported on is 1.4.1.3.1, with a Latest version of 1.4.1.3.1.2. By default, the vhistory command would only report on versions 1.4.1.3.1.1 and 1.4.1.3.1.2. If "1" was specified as the -lastbranches value, then the vhistory command would also run on one parent branch back, reporting versions 1.4.1.3, 1.4.1.2 and 1.4.1.1.

An <n> value of "all" will run the report on all parent branches, back to branch 1.

The -lastbranches option is mutually exclusive with the -descendants option. That is because specifying a -lastbranches value implies a -descendants value of 0.

The -lastbranches and -lastversions options can be used together. The report will start at the Latest version on the initial branch, and work backwards.

-lastversions

-lastversions <n>

How many versions back to report. By default, all versions on the requested branch are reported (an <n> value of "all").

You may specify any positive number as the <n> value.

The -lastversions option is mutually exclusive with the -descendants option. That is because specifying a -lastversions value implies a -descendants value of 0.

If a specific version object URL is specified as the argument (or -modulecontext), instead of a -branch, then the report will start at the version specified. (Instead of starting at the Latest version on the branch.) This allows the report to be run on a range of versions.

The -lastversions and -lastbranches options can be used together. The report will start at the Latest version on the initial branch, and work backwards.

-maxtags

-maxtags <n>

The maximum number of tags shown for any object. By default, all tags are shown (an <n> value of "all").

-modulecontext (Module-based)

```
-modulecontext <context>
              Specifies the module context. Use this option to
              identify a module member that is not in the workspace
             or to restrict the report to module versions that
             affect any of the members specified on the command line.
```

-output

```
-output <filename>
              Prints results to the specified file. The named file is
              created or overwritten, but not appended to. To append,
              use the "-stream" option.

              The -output and -stream options are mutually exclusive.
```

-[no]recursive (File-based)

```
-[no]recursive
               For a local folder or server folder, whether to
              descend through sub-folders of the starting folder, or
              only report on the objects in the specified  folder.

               The default behavior is "-norecursive".
```

-report

```
-report <mode>
               Specifies what information about each object should
               be reported. Available report modes are:

               brief    Report tagged versions/branches with their
                        tags and numerics. This is equivalent to
                        "-report NBRIVT".

               normal   Report all available information, except for
                        the module manifest. This is equivalent to
                        "-report" with all codes listed in the
                        Report Options table above, except for GZQI.

                        This behavior is the default when "-report"
                        is not specified.

               verbose  Report all available information. This is
                        equivalent to "-report" with all codes listed
                        in the Report Options table above, except for
                        GZI.

               K[K...]  Display the fields corresponding to the data
                        keys, where K is a data key listed in the
```

```
                    Report Options table above.

                  You may also use "+" and "-" operators to add and
                  remove codes from the standard reports.

                  For example, to report the "normal" output, but only
                  for version objects and not branch objects:

                  stcl> vhistory -report normal-B

                  The data keys and predefined report modes may be
                  combined in any order. However, the predefined report
                  mode names may not be immediately preceded or followed
                  by another data key or predefined report name.

                  For example, the following is valid:

                  stcl> vhistory -report Z+normal-B

                  The above command will report the "normal" output, but
                  without branches, and with the versions in reverse
                  order.

                  The following syntax is not valid:

                  stcl> vhistory -report Znormal-B

                  If the "-report" value begins with a "+" or "-", the
                  default "normal" predefined report is automatically
                  prepended.

                  For example:

                  stcl> vhistory -report -B

                  is equivalent to:

                  stcl> vhistory -report normal-B
```

-[no]selected

```
 -[no]selected
                  Whether to operate on the items in the select list in
                  addition to any arguments on the command line. If no
                  arguments are given on the command line, then the
                  select list is automatically used.
```

-stream

```
 -stream <port>
                Prints results to the specified named Tcl port.
                Depending on whether you open the stream using the Tcl
```

```
                "open" command in write (w) or append (a) mode, you can
                overwrite or append to an existing file.

                Note: The -stream option is only applicable in the stcl
                      and stclc shells, not in the dss and dssc shells.

                The -stream and -output options are mutually exclusive.
```

-xtras (Module-based)

```
 -xtras <list>        List of command line options to pass to the
                      external module change management system. Any
                      options specified with the -xtras option are
                      sent verbatim, with no processing by the
                      populate command, to the Tcl script that defines
                      the external module change management system.
```

--

```
 --                   Indicates that the command should stop looking for
                      command options. Use this option when arguments
                      to the command begin with a hyphen (-).
```

RETURN VALUE

```
If "-format list" was specified, and neither the "-output" option nor
the "-stream" option were specified, then the result list is returned.
Otherwise, a value of "" is returned.
```

SEE ALSO

```
command defaults, datasheet, ls, select, whereused member,
vhistory-foreach, vhistory-foreach-obj
```

EXAMPLES

- Example of Version History of a Module Branch (Module-based)
- Example of Version History Showing Module Rollback Operation (Module-based)
- Example of Vhistory Showing a Retired Branch (File-based)

Example of Version History of a Module Branch (Module-based)

```
The example below shows the default "-report normal" output, for a
module branch:
```

```
stcl> vhistory -branch Silver sync://faure:30044/Modules/multiple/M1
Object:           sync://faure:30044/Modules/multiple/M1
---------------------------
Branch:           1.5.1
Branch tags:      Silver
Tag comments:
    Silver : Branching version 1.5
Comment:          Branching version 1.5

---------------------------
Version:          1.5.1.1
Derived from:     1.5
Date:             Thu Oct 12 16:35:23 EDT 2006
Author:           mark
Comment:          Branching version 1.5

---------------------------
Version:          1.5.1.2
Version tags:     Latest
Derived from:     1.5.1.1
Date:             Thu Oct 12 16:36:41 EDT 2006
Author:           mark
Comment:          Versioning new silver branch

============================================================================
stcl>
```

The example below shows "-report verbose" output, for a module branch:

```
stcl> vhistory -branch Golden sync://faure:30044/Modules/multiple/M1 \
-report verbose
Object:           sync://faure:30044/Modules/multiple/M1
---------------------------
Branch:           1.9.1
Branch tags:      Golden

---------------------------
Version:          1.9.1.1
Derived from:     1.9
Date:             Thu Oct 12 16:28:43 EDT 2006
Author:           debra
Manifest:
    Added : /m1/c.txt, 1.1
    Added : /unixfilesfolder/unixfile1.txt, 1.2
    Added : /1.txt, 1.2
    Added : /m1/d.txt, 1.1
    Added : /m1/a.txt, 1.3
    Added : /m1/b.txt, 1.1

---------------------------
Version:          1.9.1.2
Version tags:     Latest
Derived from:     1.9.1.1
Date:             Thu Oct 12 16:31:13 EDT 2006
Author:           debra
Comment:          Testing some changes.
Manifest:
```

```
    Added : /unixfilesfolder,
    Added : /file2, 1.1
    Changed : /1.txt, 1.2 -> 1.2.1.1
    Added : /file1, 1.1
    Added : /file3, 1.1
    Added : /m1,
    Renamed : /m1/c.txt -> /m1/x.txt, 1.1
    Deleted: /1.txt
    Renamed,Changed : /m1/a.txt -> /m1/ab.txt, 1.3 -> 1.4
=============================================================================
stcl>
```

Example of Version History Showing Module Rollback Operation (Module-based)

```
This example shows a Module rollback operation in which version 1.5
of the MBOM module was created by rolling back to version 1.2,
removing the changes introduced in version in version 1.3 and 1.4.
The example vhistory output includes a graphical representation
(-report G).

Note:  The rollback comment is displayed as the checkin comment for
the module version created from the rollback.

dss> vhistory -norecursive -report NSWLTYDFAECXUHVBRG MBOM%0
Object:           /home/rsmith/MyModules/mbom/MBOM%0
Vault URL:        sync://srv2.ABCo.com:2647/Modules/MBOM
Current version:  1.4
----------------------------
Branch:           1
Branch tags:
  Trunk, Wed Sep 05 08:20:54 AM EDT 2007, rsmith

----------------------------
Version:          1.1
Date:             Wed Sep 05 08:20:55 AM EDT 2007
Author:           rsmith
Comment:          First Version

----------------------------
Version:          1.2
Derived from:     1.1
Date:             Wed Sep 05 08:24:34 AM EDT 2007
Author:           rsmith
Comment:          Initial checkin

----------------------------
Version:          1.3
Derived from:     1.2
Date:             Wed Sep 05 08:26:21 AM EDT 2007
Author:           rsmith
Comment:          Updates to documentation and base code.

 ----------------------------
 Version:          1.4
```

```
 Derived from:     1.3
 Date:             Wed Sep 05 08:26:44 AM EDT 2007
 Author:           rsmith
 Comment:          added header file

----------------------------
Version:          1.5
Version tags:     Latest
Derived from:     1.4
Merged from:      1.2
Date:             Wed Sep 05 08:35:00 AM EDT 2007
Author:           rsmith
Comment:          introduced incompatable changes

----------------------------
History Graph:

1 (Trunk)
1.1
1.2 => 1.5
1.3
1.4
1.5 [Latest] <= 1.2
============================================================================
```

Example of Vhistory Showing a Retired Branch (File-based)

```
This example shows a retired branch. The report mode used is normal,
which includes the report options: B and X.  Both options are
required to see the time, date and username associated with the
retire.  The first command output shows the default format, text.
The second shows the -format list output.

dss> vhistory c.doc
 Object:           file:///home/rsmith/workspaces/M1/doc/c.doc
 Vault URL:        sync://srv2.ABCo.com:2647/Projects/M1/doc/c.doc;
 Current state:    NotFetched (Locally Modified)
 ----------------------------
 Branch:           1
 Branch tags:      Trunk
 This branch is retired.
 Retired by bjones on Tue Dec 30 01:48:48 PM EST 2008

 ----------------------------
 Version:          1.1
 Version tags:     Latest
 Date:             Tue Dec 30 01:39:16 PM EST 2008
 Author:           rsmith
 Comment:          Updates to documentation

 ============================================================================

dss> vhistory -format list c.doc
 {name file:///home/rsmith/workspaces/M1/doc/c.doc url
```

```
{sync://srv2.ABCo.com:2647/Projects/M1/doc/c.doc;} state {NotFetched
(Locally Modified)} objects {{type branch version 1 bud 0 tags Trunk
tag_properties {{Trunk 0 {} {} {}}} locker {} upcoming {} retired 1
retired_properties {date 1230662928 user bjones} comment {}} {type
version version 1.1 tags Latest tag_properties {{Latest 0 {} {} {}}}
derived_from {} merged_from {} date 1230662356 author rsmith comment
{Updates to documentation}}}}
```

vhistory-foreach

vhistory-foreach Command

NAME

```
vhistory-foreach    - Function to process the results of a vhistory
                      command
```

DESCRIPTION

```
This function is called on the result list returned from
"vhistory -format list". Use the vhistory-foreach function in
conjunction with the vhistory-foreach-obj function, to process the
list result from vhistory.
```

SYNOPSIS

```
vhistory-foreach obj result_list <tcl_script>
```

ARGUMENTS

- Object Loop Variable
- Results List
- Tcl Script

Object Loop Variable

```
obj                 This is the loop variable. It is treated as a Tcl
                    array. The "obj" Tcl array is set to each object
                    in the result list, in turn.

                    The Tcl array contains the properties for the
                    object, and an "objects" property containing the
                    version and branch entries that were reported for
                    the object.
```

The set of properties is determined by the "-report" option that was specified to the "vhistory" command. If a "-report" value is not specified, the default "normal" report keys are used.

Results List

result_list — The result list to be processed. This is the result value from a call to the "vhistory" command with the "-format list" option.

Tcl Script

tcl_script — The Tcl code to execute on each element in the "obj" Tcl array.

SEE ALSO

vhistory, vhistory-foreach-obj

EXAMPLE

As an example, let's use the vhistory report from the "-format list" option description in the "vhistory" command documentation:

```
stcl> vhistory -report LNRVT file1.txt file2.txt -last 2 -format list
```

We'll capture the result in a variable, then use the vhistory-foreach functions to process the result:

```
set result [vhistory file1.txt file2.txt -lastversions 2 -format list]

vhistory-foreach obj $result {
    puts "Object name: $obj(name)"

    vhistory-foreach-obj vb obj {
        if { $vb(type) == "version" } {
                puts "Version: $vb(version)"
        } else {
                puts "Branch: $vb(version)"
        }
    }
}
```

The above code would report:

```
Object name: file1.txt
Version: 1.4
Version 1.5
Object name: file2.txt
Version: 1.3.1.5
Version 1.3.1.6
```

vhistory-foreach-obj

vhistory-foreach-obj Command

NAME

```
vhistory-foreach-obj- Function to process the results of a vhistory
                      command
```

DESCRIPTION

```
This function is called with the property array that was set by the
vhistory-foreach function. The two vhistory "foreach" functions are
used to process the list result from vhistory.
```

SYNOPSIS

```
vhistory-foreach-obj vb obj <tcl_script>
```

ARGUMENTS

- Version/Branch Loop Variable
- Object Tcl Array
- Tcl Code

Version/Branch Loop Variable

```
vb                  The version/branch entry. This is the loop
                    variable. The "vb" Tcl array is set to each
                    version or branch entry for the object, in turn.
```

Object Tcl Array

```
obj                 This is the "obj" Tcl array that was set by the
```

```
                    "vhistory-foreach" function.
```

Tcl Code

```
tcl_script          The Tcl code to execute on each element in the
                    "vb" Tcl array.
```

SEE ALSO

```
vhistory, vhistory-foreach
```

EXAMPLE

```
As an example, let's use the vhistory report from the "-format list"
option description in the "vhistory" command documentation:

stcl> vhistory -report LNRVT file1.txt file2.txt -last 2 -format list

We'll capture the result in a variable, then use the vhistory-foreach
functions to process the result:

set result [vhistory file1.txt file2.txt -lastversions 2 -format list]

vhistory-foreach obj $result {
    puts "Object name: $obj(name)"

    vhistory-foreach-obj vb obj {
        if { $vb(type) == "version" } {
                puts "Version: $vb(version)"
        } else {
                puts "Branch: $vb(version)"
        }
    }
}

The above code would report:

Object name: file1.txt
Version: 1.4
Version 1.5
Object name: file2.txt
Version: 1.3.1.5
Version 1.3.1.6
```

webhelp

webhelp Command

NAME

```
webhelp             - Launches Graphical Web Browser to view help
```

DESCRIPTION

```
This command provides a variety of help related functions, displaying
the information in the default web browser.  The default web browser
is set during DesignSync client installation. You can change or set
the web browser at any time using SyncAdmin. For more information on
setting the web browser, see the ENOVIA Synchronicity DesignSync Data
Manager Administrator's Guide.

Help is available for:
  - All DesignSync command-line commands
  - DesignSync topics such as using wildcards or running server-side
    scripts
  - ProjectSync command-line commands

For compound commands such as the 'url' and 'note'
commands, surround the command with double quotes and put
exactly one space between the two keywords of the command (see
Example section).

The web browser opens the specified help topic within the ENOVIA
Synchronicity Command Reference for the selected help mode you are
working in.  For information about setting a help mode, see the
ENOVIA Synchronicity DesignSync Data Manager Administrator's Guide.
By default, the help mode is "all," which includes the DesignSync
documentation for all working modes, including modules, files-based,
and legacy modules modes. You can also specify a help mode using the
-mode option.

From the ENOVIA Synchronicity Command Reference, you can navigate to
the documentation index to access any other DesignSync documentation.
```

SYNOPSIS

```
webhelp [-mode module|file|all] [<topic> [...]]
```

ARGUMENT

- Topic

Topic

```
<topic>[...]    DesignSync command name(s) or topic(s).
                If the topic or command specified doesn't exist,
```

```
the webhelp command launches the web browser and
displays the overview topic.

If you specify more than one topic, each topic will
open in a separate tab in the web browser.

Note: When looking up a two word topic, such as
"defaults show" enclose the command in quotes,
otherwise it will be processed as two separate
topics.  In this example, entering the command
"webhelp defaults show" would result in two tabs
being opened, one to the "defaults" topic and one to
the overview page, since there is no corresponding
"show" command.
```

OPTIONS

- -mode

-mode

```
-mode module|  Determines which version of the help to open.
 file | all    If you specify the -mode option, the setting you
               choose overrides the default mode.

               If no mode is specified, DesignSync uses the default
               mode defined with the registry key or SyncAdmin. For
               more information on defining the help mode, see the
               ENOVIA Synchronicity DesignSync Data Manager
               Administrator's Guide.  If no mode is set, the help
               page displays in the "all" mode.

               Note: Once the book is open, you can navigate to the
               documentation index and from there open a different
               version of the ENOVIA Synchronicity Command Reference.
```

RETURN VALUE

```
If the command succeeds, it returns an empty string (""). If the
command fails, it returns an appropriate error to explain the cause
of failure.
```

EXAMPLES

- Example of Opening a Single Tab in the Default Mode
- Example of Opening Multiple Tab Help for a Specified Mode (Module-based)
- Example of Opening Multiple Tab Help for a Specified Mode (File-based)

Example of Opening a Single Tab in the Default Mode

The following example opens one tab to the "default show" command.

```
dss> webhelp "defaults show"
```

Note: The quotes are required because the command is more than a single word.

Example of Opening Multiple Tab Help for a Specified Mode (Module-based)

The following example opens two tabs in the specified user mode "module." Using a help mode ensure that all the information provided is specific to the data management methodology you are using.

```
dss> webhelp -mode module addhref edithrefs
```

Example of Opening Multiple Tab Help for a Specified Mode (File-based)

The following example opens two tabs in the specified user mode "file." Using a help mode ensure that all the information provided is specific to the data management methodology you are using.

```
dss> webhelp -mode file ls ci
```

Enterprise Design Development

Development Areas

sda

sda Command

```
sda                  - Synchronicity development area commands
```

DESCRIPTION

```
The sda commands allow you to manage your DesignSync development
areas. For more information on development areas, see the
Enterprise Design Administration User's Guide.

Note: The sda commands must be run from your OS shell, not from
      within the DesignSync interfaces.
```

SYNOPSIS

```
sda <sub_command> [<sub_command_options>]

Usage: sda [cd|gui|join|ls|mk|rm]
```

OPTIONS

```
Vary by command.
```

RETURN VALUE

```
Varies by command.
```

SEE ALSO

```
sda cd, sda gui, sda join, sda ls, sda mk, sda rm
```

EXAMPLES

See specific "sda" commands.

sda cd

sda cd Command

NAME

sda cd - Change development area and launch a tool command

DESCRIPTION

This command allows the user to launch a tool from a development area they have created via "sda mk" or joined via "sda join". The tool runs using the development setting defined for the area.

The sda cd command performs the following sequence of actions:

1. If the -update option is selected, updates the development instance directory associated with an external development area.

2. Sets up the environment by setting the following environment variables:
   - SYNC_DEVAREA_DIR - set to the requested development area directory.
   - SYNC_DEVAREA_TOP - set to the leaf name of the top module or directory in the development area.
   - SYNC_DEV_ASSIGNMENT - set to the assignment associated with the development area.
   - SYNC_DEVELOPMENT_DIR - set to the top of the development instance directory.
   - SYNC_PROJECT_CFGDIR - set to the directory holding the development setting for the assignment associated with the development area.
   - SYNC_WS_DEVAREA_TOP - set to the leaf name of the top module or directory in the development area. This variable can then be used for the starting directory in any commands you construct within the specified tool.

3. Runs all of the set up scripts defined for the tools associated with the development area. Running all the scripts is required to support inter-tool dependencies and shell tools.
   Note: When a shell is defined as a tool, it should be defined to ignore the startup script for the shell. Any aliases, etc. defined in the startup script will not be available; however when a tool suite is defined, the admin can specify a script with the desired environment settings.

4. Sets the current directory for the tool to the starting directory. The starting directory is the directory defined in the tool's definition. If no starting directory is specified, then the directory defined in the tool suite is used. If no starting

directory is specified in the tool suite either, the development area is used.
The starting directories can be specified with environment variables and may be relative to the development area.

5. Starts the requested tool. If the tool is graphical, the tool is spawned (detached) from sda. If the tool is non-graphical, on UNIX, the tool runs in the same shell as sda.

   Note: When a non-graphical tool is started, the sda process ends.

If you run the command without specifying a development area or a tool, or the user specified an ambiguous argument, the command starts in interactive mode. In interactive mode, the user is prompted for the command arguments and options needed. Any arguments specified with the -gui command option are passed to the GUI and the appropriate fields are selected on the "Change Area" tab.

SYNOPSIS

```
sda cd [<area_name>] [<tool>] [-development <name>] [-gui]
       [-suite <suite_name>] [-[no]update] [-version <version>]
```

ARGUMENTS

- Development Area Name
- Tool

Development Area Name

area_name	The development area name of the DesignSync Development. This argument is required and the development area must already exist.

Tool

tool	The tool name specified must be a tool that is defined for use with the specified development area. The list of available tools can be viewed from the development instance for the assignment associated with the area. Note: When a shell is defined as a tool, it should be defined to ignore the startup script for the shell. Any aliases, etc. defined in the startup script will not be available.

OPTIONS

-development

-development <name>	Specify the name of the development if the area name is not unique for the user. Area names are unique within a development for a given user, but are not required to be unique across all developments.

-gui

gui	Starts the sda graphical user interface mode with the "Change Area" tab selected. If this option is used with the tool argument, the tool argument is silently ignored.

-suite

-suite <suite>	Specify the suite name for the tool suite, if the tool name is not unique across all tool suites for the development assignment.

-[no]update

-[no]update	Specifies whether the development instance definition should be updated, if it is an external area. -noupdate does not update the external development instance from the server before setting the environment variables for the area and starting the tool. (Default when the development setting is 'Mirror=False') -update performs the update of the external area before performing any other actions. (Default when 'Mirror=True') If the area is not an external area and this option is specified, the tool exits without launching the tool. Note: If -update is explicitly specified, and no

```
                        tool is specified, DesignSync assumes the
                        desired action is the update and does not
                        prompt for tool in interactive mode.
```

-version

```
 -version          Specify the version number of the tool suite if the
  <version>        tool suite name is not unique within the
                   development assignment. This option must be
                  specified if there are multiple tools with the same
                  name in multiple tool suites with the same name.
```

RETURN VALUE

```
There is no TCL return value for this command.
```

SEE ALSO

```
sda gui, sda join, sda ls, sda mk, sda rm
```

EXAMPLES

- Running sda cd in Interactive Mode
- Running sda cd in non-interactive mode

Running sda cd in Interactive Mode

```
This example runs sda cd in interative mode, supplying no
arguments. It is run from a Windows client and launches the
DesignSync GUI which is configured as a tool for this development
area.
Note that the list of areas is prefixed with the development name for
ease of idenfitication.

C:\workspaces\chipNZ214> sda cd
Logging to C:\Users\fyl\dss_11042013_100431.log
V6R2014x

Which development area would you like to work with?
[1] (Chip-NZ214) documenter-1_rmsith
[2] (Chip-QR2) verifier-1_thopkins
[3] (Chip-NZ214) developer-1_rsmith
[E] <EXIT sda>
Select the number preceding the development area name or 'E' to exit
 [1-3,E]: 1
```

```
Synchronizing the local development with the server ...
Contacting host: serv1.ABCo.com:2164 ...
Synchronization complete

Which tool would you like to launch?
[1] Authoring Tool
[2] DesSync
[E] <EXIT sda>
Select the number preceding the tool name or 'E' to exit (1-2,E): 2

c:\workspaces\chipNZ214>
```

Running sda cd in non-interactive mode

```
This example specifies the area and tool and the -noupdate option.
Note that it does not enter interactive mode, nor does it attempt to
synchronize the development area.  This example automatically
launches the GUI tool, without requiring the -GUI option because of
the way the tool is defined.

C:\workspaces\chipNZ214> sda cd Chip-NZ214 DesSync -noupdate
Logging to C:\Users\fyl\dss_11042013_103110.log
V6R2014x
[The DesignSync Development Area Manager launches in separate window]
c:\workspaces\chipNZ214>
```

sda gui

sda gui Command

NAME

```
sda gui             - Start the sda area management graphical user
                      interface
```

DESCRIPTION

```
This command is used to start the graphical user interface
sda tool. The sda GUI tool is a tabbed dialog based tool for
development area management. When the GUI is opened from this
command, it displays the most recently used tab.

For information on using the sda GUI tool, see the Enterprise DesignSync
Administration User's Guide

Note: If you are running this from UNIX, you must use an environment that
supports running graphical clients.
```

SYNOPSIS

```
sda gui
```

RETURN VALUE

```
This command has no TCL return value.  If the GUI is unable to
launch, the command returns an appropriate error message.
```

SEE ALSO

```
sda cd, sda join, sda ls, sda mk, sda rm
```

EXAMPLES

- Starting sda GUI in the Background

Starting sda GUI in the Background

```
This example starts the sda GUI as a background process on UNIX,
leaving the terminal free to type additional commands if needed.

> sda GUI &
```

sda join

sda join Command

NAME

```
sda join            - Allow the user to join an existing development
                      area
```

DESCRIPTION

```
This command allows the user to join an existing eligible shared
area of a development. Eligible shared areas are located by
finding the participating development servers, looking at the
developments on those servers and identifying the shared areas that
have a local path and have not already been joined. For information
on defining a development server, see the DesignSync Data Manager
Administrator's Guide.
```

If you don't specify arguments to sda join, it starts in interactive mode, prompting you for any information needed that was not provided on the command line.

SYNOPSIS

```
sda join [<area_name>] [-development <name>] [-gui]
```

ARGUMENTS

- Area Name

Area Name

area_name	The name of the DesignSync development area. The area must already exist. If the area is not provided, or cannot be uniquely identified from the name, you are prompted for the area name in interactive mode. If an invalid area is specified, and the -gui option is used, the GUI starts on the "Join Area" tab and allows you to select a valid Area.

OPTIONS

- -development
- -gui

-development

-development <name>	Specify the development name if the area name is not unique for the user. Area names are unique within a development for a given user, but are not required to be unique across all developments.

-gui

-gui	Starts the sda graphical user interface mode with the "Join Area" tab selected.

RETURN VALUE

```
There is no TCL return value for this command. If the command fails,
DesignSync returns an appropriate error.
```

SEE ALSO

```
sda cd, sda gui, sda ls, sda mk, sda rm
```

EXAMPLES

sda ls

sda ls Command

NAME

```
sda ls              - List the areas or developments relevant to the
                      user
```

DESCRIPTION

```
This command lists the areas or developments that are currently
active for the user, or registered with the development servers
defined in SyncAdmin.  For more information on defining development
servers, see the DesignSync Data Manager Administrator's Guide.
```

SYNOPSIS

```
sda ls [-area | -development] [-gui] [-noheader]
       [-report brief | normal | verbose]
```

OPTIONS

- -area
- -development
- -gui
- -noheader
- -report

-area

-area	Show all of the areas, sorted by name, that are currently active for the user.

-development

-development	Show all the developments available from the development servers associated with the distribution. Development servers are associated with a distribution using SyncAdmin.

-gui

-gui	Starts the sda graphical user interface mode with the "List Areas," or "List Developments" tab selected.

-noheader

-noheader	Specifies omitting column headers for the command line reports. This option is silently ignored when the -gui option is specified. If this option is not specified, the command line reports include column headers.

-report

-report brief|normal|verbose	Specifies the amount of output supplied by the command. When -area is specified: -report brief - lists area names. -report normal - lists the area name, development name, and assignment associated with the area. -report verbose - includes all the information from -report normal and the path to the area directory, development's local instance directory, and status (enabled, disabled, or deleted.) When -development is specified: -report brief - lists development names. -report normal - lists the development name and its supported assignments. -report verbose - includes all the information in -report normal and the data URL, selector, development path, server URL, and status (enabled, disabled, or deleted.)

RETURN VALUE

```
This command does not return any TCL values. If the command succeeds,
it displays the list of development areas.  If the command fails, it
fails with an appropriate error.
```

SEE ALSO

```
sda cd, sda gui, sda join, sda mk, sda rm
```

EXAMPLES

- Example Showing the List of Development Areas

Example Showing the List of Development Areas

```
This example shows a list of the defined development areas. Note that
this command runs at the shell, not in the dss/stcl environment.

$> sda ls
Logging to /home/rsmith/dss_08112016_103913.log
3DEXPERIENCER2021x

Development   Development Area    Assignment
-----------   ----------------    ----------
ChipNZ214     documenter-1_rsmith QATester
```

sda rm

sda rm Command

NAME

```
sda rm              - Remove an existing development area and its
                      contents
```

DESCRIPTION

```
This command removes a development area from its development
definition on the development server and attempts to remove the
local development area directory. If the development area is a shared
development area, only the last user to remove the development area
is allowed to remove the local development area directory. The
```

```
command does not remove any design data from the repository server.

Invoking sda rm without any arguments, or with incomplete or
ambiguous arguments, causes the command to enter the interactive
mode. In interactive mode, the user is prompted for the command
arguments and options needed and must confirm the answers.

In interactive mode, orphaned development areas, development areas
where a development instance can't be found;  are displayed
preceded with a "!" and shared development areas are displayed
preceded with a "*".

Any arguments specified with the -gui command option are passed to
the GUI and the appropriate fields are pre-filled on the Remove Area
tab. The GUI ignores the -noconfirm option if it is used.
```

SYNOPSIS

```
sda rm [<area_name>] [-development <name>] [-gui] [-noconfirm]
```

OPTIONS

- -development
- -gui
- -noconfirm

-development

```
-development      Specify the development name if the area name is
 <name>           not unique for the user. Area names are unique
                 within a development for a given user, but are not
                 required to be unique across all developments.
```

-gui

```
-gui              Starts the sda graphical user interface mode
                  with the "Remove Area," tab selected.
```

-noconfirm

```
-noconfirm         By default, the removal requires confirmation.  Use
                   the -noconfirm option to perform the removal
                  without confirmation.

                   Note: The GUI interface always requires
                   confirmation.  If -noconfirm is specified with
```

```
                    -gui, the -noconfirm option is silently ignored.
```

RETURN VALUE

```
This command does not return any TCL values.  The command output displays
information about success or failure of the command and status messages.
```

SEE ALSO

```
sda cd, sda gui, sda join, sda ls, sda mk
```

EXAMPLES

Example Showing Removing a Development

```
$> sda rm nz8ChipDev
** You are removing both the development area definition and the
development area directory. **
Are you sure you want the remove development area 'nz8ChipDev' from
development 'Chip-NZ8'? (y/n) [n]:y
You have successfully removed development area 'nz8ChipDev' from
development 'Chip-NZ8'.
$>
```

Example Showing Removing a Development in Interactive Mode

```
$> sda rm
Which development area would you like to remove?
[1] nz8ChipDev (Chip-NZ8)
[2] nz8ChipDev (ROM-NZx)
[3] Chip-QR2
[4] ROM-NZx
* Shared development area
Select the number preceding the development area name (1-5):1

** You are removing both the development area definition and the
directory. **
Are you sure you want the remove development area 'nz8ChipDev' from
development 'Chip-NZ8'? (y/n) [n]:y
You have successfully removed development area 'nz8ChipDev' from
development 'Chip-NZ8'.
```

Enterprise Object Viewing and Synchronization

entobj

entobj Command

NAME

```
entobj           - Commands to work with Enterprise Design Objects
```

DESCRIPTION

```
These commands provide a link between the Enterprise Design Objects
and associated DesignSync objects. These commands are available from
the DesignSync client and the server.

For information about the specific entobj commands including
arguments, options, and examples to support Enterprise Design
objects in DesignSync, see the individual command descriptions.
```

SYNOPSIS

```
entobj <entobj_command> [entobj_command options>]

Usage: entobj [id | isplatformmanaged | policy | setpolicy | settype
| show | synchronize | type]
```

entobj id

entobj id Command

NAME

```
entobj id          - Returns the platform identifier
```

DESCRIPTION

```
Returns the platform identifier for the object. The platform
identifier is the unique string for the object on the Enterprise
system that is associated with the DesignSync object.

This command is subject to access controls on the server.
```

SYNOPSIS

```
entobj id <argument>
```

ARGUMENTS

- Server URL

Server URL

```
<serverURL>         Specifies the URL of the module, module version,
                    or module branch version.

                    Note: To get the ID associated with a module
                    branch, you must specify the branch numeric value.

                    Specify the URL as
                    follows:
  sync://<host>[:<port>]/Modules/[<category...>/]<module>;[<selector>]
                    or
  syncs://<host>[:<port>]/Modules/[<category...>/]<module>;[<selector>]
                    where 'sync://' or 'syncs://' is required, <host>
                    is the machine on which the SyncServer is
                    installed, <port> is the SyncServer port number
                    (defaults to 2647/2679), [<category...>} is the
                    optional category (and/or sub-category) containing
                    the module, and <module> is the name of the
                    module. For example:
              sync://serv1.abco.com:1024/Modules/ChipDesigns/Chip

                     Note: When a selector is not provided, the
                     default, Trunk:Latest, is used.

                   To get the ID associated with a module branch, you
                   must specify the branch numeric value.
```

RETURN VALUE

```
Returns the unique identifier for the object. If an argument is
specified that doesn't have an ID or is not a module, returns an empty
value ("").
```

SEE ALSO

```
entobj isplatformmanaged, entobj show
```

EXAMPLES

- Example of a request for the id

Example of a request for the id

```
This example shows the ID for the Enterprise Design managed
DesignSync object.

dss> entobj id sync://serv1.ABCo.com:2647/Modules/CPU
2341B75697660000EBA9F35673620700
```

entobj isplatformmanaged

entobj isplatformmanaged Command

NAME

```
entobj isplatformmanaged - Retuns whether the object is managed by
                           the enterprise system.
```

DESCRIPTION

```
This command tells you if the DesignSync object is managed by the
Enterprise Design system; meaning that the object was created as a
result of pushing a definition down from the Enterprise Design
system, rather than being reflected from DesignSync to the system.

This command is subject to access controls on the server.
```

SYNOPSIS

```
entobj isplatformmanaged <argument>
```

ARGUMENTS

- Server URL

Server URL

```
<serverURL>         Specifies the URL of the module, module version,
                    or module branch version.

                    Specify the URL as follows:
 sync://<host>[:<port>]/Modules/[<category...>/]<module>;[<selector>]
                    or
 syncs://<host>[:<port>]/Modules/[<category...>/]<module>;[<selector>]
                    where 'sync://' or 'syncs://' is required, <host>
```

```
                is the machine on which the SyncServer is
                installed, <port> is the SyncServer port number
                (defaults to 2647/2679), [<category...>} is the
                optional category (and/or sub-category) containing
                the module, and <module> is the name of the
                module. For example:
        sync://serv1.abco.com:1024/Modules/ChipDesigns/Chip

                 Note: When a selector is not provided, the
                 default, Trunk:Latest, is used.
```

RETURN VALUE

```
Command returns "1" if the object is managed from the platform or "0"
if the object is not managed from the platform. Command returns an
empty string (""), if an illegal argument was provided to the command.
```

SEE ALSO

```
entobj id, entobj show
```

EXAMPLES

- Example Showing That an Object is Managed from the Enterprise System
- Example Showing That an Object is Not Managed by the Enterprise System

Example Showing That an Object is Managed from the Enterprise System

```
This example shows the reply from the system when an object is
managed from the Enterprise System.

dss> entobj isplatformmanaged sync://serv1.ABCo.com/Modules/CPU;Gold
1
```

Example Showing That an Object is Not Managed by the Enterprise System

```
This example shows the reply from the system when an object is
not managed by the Enterprise System.

dss> entobj isplatformmanaged sync://serv1.ABCo.com/Modules/ROM;Gold
0
```

entobj policy

entobj policy Command

NAME

```
entobj policy       - Displays the assigned policy for the module
```

DESCRIPTION

```
This command displays the policy for the module.  If there is no
policy set for the module, the command returns a null value ("").

This command is subject to access controls on the server.
```

SYNOPSIS

```
entobj policy <module>
```

ARGUMENTS

- Module instance
- Server URL

Module instance

```
<Workspace Module>  The workspace module instance name; for example:
                    Chip%0.
```

Server URL

```
<Server URL>        The Server module URL in the format:
       sync[s]://<host>[:<port>]/Module/[<category>...]/<ModuleName>
                    where <host> is the SyncServer on which the module
                    resides, <port> is the SyncServer port number,
                    <category> identifies path to the module, and
                    <ModuleName> is the module associated with the
                    Enterprise Design object.
```

RETURN VALUE

```
This command does not return any TCL values. When successful, this
command returns the policy name or, if no policy is assigned, an
empty string (""). If the command fails, DesignSync displays an error
message explaining the failure.
```

SEE ALSO

```
 entobj setpolicy
,
```

EXAMPLES

- Example of Showing the Set Policy

Example of Showing the Set Policy

```
 dss> entobj setpolicy sync://serv2.ABCo.com:2647/Modules/ChipDZ
 QualityTest
```

entobj setpolicy

entobj setpolicy Command

NAME

```
 entobj setpolicy    - Set or remove the policy for a module.
```

DESCRIPTION

```
 This command assigns or removes a product policy for a module. The
 policy is used on the Enterprise platform.  DesignSync does not
 perform any validation on the policy to determine if the policy is in
 use on the Enterprise platform.  The policy is case-sensitive.

 Tip: Before applying a new policy, synchronize the module to verify
 that the module has the latest information from the platform.  If the
 product ID has changed since the last synchronization, you cannot
 set the policy.

 The policy set by this command is synchronized with the object on the
 enterprise server when the next automatic or manual synchronization
 is performed.

 This command is subject to access controls on the server.
```

SYNOPSIS

```
 entobj setpolicy  <module> <policy>
```

ARGUMENTS

- Module instance
- Server URL

Module instance

```
<Workspace Module>  The workspace module instance name; for example:
                    Chip%0.
```

Server URL

```
<Server URL>        The Server module URL in the format:
       sync[s]://<host>[:<port>]/Module/[<category>...]/<ModuleName>
                    where <host> is the SyncServer on which the module
                    resides, <port> is the SyncServer port number,
                    <category> identifies path to the module, and
                    <ModuleName> is the module associated with the
                    Enterprise Design object.
```

OPTIONS

```
<policy>            Policy for the Enterprise development.  The
                    policy is case-sensitive, and in order to be used
                    must be identify toe the policy on the Enterprise
                    system.

                    The entobj setpolicy command does not perform any
                    validation on the policy name.
```

RETURN VALUE

```
If this command is successful, it returns a null string ("").  If the
command is unsuccessful, it returns an appropriate error explaining
the failure.
```

SEE ALSO

```
entobj policy, entobj settype, entobj synchronize, entobj type
```

EXAMPLES

- Example of Setting the Policy on Enterprise Development Module
- Example of Removing the Policy on an Enterprise Development Module

Example of Setting the Policy on Enterprise Development Module

```
This example shows setting the policy for an enterprise development
module.

dss> entobj setpolicy sync://serv2.ABCo.com:2647/Modules/ChipDZ \
QualityTest
```

Example of Removing the Policy on an Enterprise Development Module

```
This example shows removing the policy for an enterprise development
module.
dss> entobj setpolicy sync://serv2.ABCo.com:2647/Modules/ChipDZ \
""
```

entobj settype

entobj settype Command

NAME

```
entobj settype      - Set the product type for the module.
```

DESCRIPTION

```
This commands sets or removes the product type for the module. The
product type should match either the user-friendly form of a product
type or the symbolic form (for example: "Software Product" or
"type_SoftwareProduct") in use on the enterprise platform.

The product type is case sensitive.

Tip: Before applying a new product type, synchronize the module to
verify that the module has the latest information from the platform.
If the product ID has changed since the last synchronization, you
cannot change the product type.

The product type set by this command is synchronized with the object
on the enterprise server when the next automatic or manual
synchronization is performed.

This command is subject to access controls on the server.
```

SYNOPSIS

```
entobj settype <module> <productType>
```

ARGUMENTS

- Module instance
- Server URL

Module instance

```
<Workspace Module>  The workspace module instance name; for example:
                    Chip%0.
```

Server URL

```
<Server URL>        The Server module URL in the format:
       sync[s]://<host>[:<port>]/Module/[<category>...]/<ModuleName>
                    where <host> is the SyncServer on which the module
                    resides, <port> is the SyncServer port number,
                    <category> identifies path to the module, and
                    <ModuleName> is the module associated with the
                    Enterprise Design object.
```

OPTIONS

```
<productType>     The name of the product type. The product type is
                  case-sensitive. The product type can be specified
                  either in a user-friendly form, such as "Software
                  Product" or in a symbolic form, such as
                  "type_SoftwareProduct".

                  To clear, or remove an associated productType,
                 specify a null string, for example, "".
```

RETURN VALUE

```
If this command is successful, it returns a null string ("").  If the
command is unsuccessful, it returns an appropriate error explaining
the failure.
```

SEE ALSO

```
entobj synchronize, entobj policy, entobj setpolicy, entobj type
```

EXAMPLES

- Example of Setting the Product Type on Enterprise Development Module
- Example of Removing the Type from an Enterprise Development Module

Example of Setting the Product Type on Enterprise Development Module

```
This example shows setting the product type for an enterprise
development module.

dss> entobj settype sync://serv2.ABCo.com:2647/Modules/ChipDZ \
"Chip Design DZ-2"
```

Example of Removing the Type from an Enterprise Development Module

```
This example shows removing the product type for an enterprise
development module.
dss> entobj settype sync://serv2.ABCo.com:2647/Modules/ChipDZ \
""
```

entobject show

entobj show Command

NAME

```
entobj show      - Show the associated enterprise server object
                   revisions
```

DESCRIPTION

```
This command shows the object revisions on an Enterprise
server associated with a DesignSync module object using the default
web browser.

For a module version or branch that has been synchronized with an
Enterprise object, this command shows the Property page for that
object in the Enterprise system.

For a workspace module instance, or a module version or branch that
has not been synchronized, this command attempt to find Enterprise
objects associated with that module version, branch, or selector and
display the results in a table.

The ENOVIA server information is stored in SyncAdmin in the Site
settings, "Enterprise Servers" tab.  For more information on
defining the ENOVIA server, see the DesignSync Data Manager
Administrator's Guide.

The ENOVIA object must have a defined DSFA connection to the
module object in DesignSync.

This command supports access controls.
```

SYNOPSIS

```
entobj show [-branch <selector>] [-version <selector>] <module>
```

ARGUMENTS

- Module instance
- Server URL

Module instance

```
<Workspace Module>  The workspace module instance name; for example:
                    Chip%0.
```

Server URL

```
<Server URL>        The Server module URL in the format:
       sync[s]://<host>[:<port>]/Module/[<category>...]/<ModuleName>
                    where <host> is the SyncServer on which the module
                    resides, <port> is the SyncServer port number,
                    <category> identifies path to the module, and
                    <ModuleName> is the module associated with the
                    Enterprise Design object.
```

OPTIONS

- -branch
- -version

-branch

```
-branch <selector>      Specifies the branch by the branch or version
                        tag, or branch numeric.

                        For a workspace module, if no -branch or
                        -version option is specified, a combination of
                        the fetched version and selector are used to
                        find matching objects in the Enterprise
                        system.

                        For a Server URL, either a -branch or a
                        -version option must be specified.

                        Note: The -branch option accepts a single
                        branch tag, a single version tag, or a branch
                        numeric. It does not accept a selector or
```

selector list.

-version

-version <selector>	Specifies the version of a module associated with the Enterprise Design objects. For a workspace module, if no -version option is selected, DesignSync uses the version fetched in the workspace and the module selector to identify the matching objects in the Enterprise system. For a server URL, you must specify either the -version or -branch options. You may specify any valid single selector. Note: You may specify a branch or version that is not among the ancestors of the branch loaded into the workspace; meaning you can unremove objects to check into the local workspace branch that were previously not present on the branch.

RETURN VALUE

This command has no TCL return value. The command launches the default web browser to display the information returned.

SEE ALSO

entobj synchronize, populate

entobject synchronize

entobj synchronize Command

NAME

entobj synchronize - Synchronize Enterprise Design Objects and DesignSync module versions and branches.

DESCRIPTION

This command synchronizes the version and hierarchy information for a DesignSync module with the corresponding Enterprise Design representation.

During synchronization, DesignSync validates ProductType and Policy, which are sent to the server in symbolic form (for example, "type_SoftwareProduct").

This command supports command defaults options.

This command is subject to access controls on the server.

SYNOPSIS

```
entobj synchronize [-depth all|one|none] [-dryrun]
[-report brief|normal|verbose] -tags <taglist> -xtras <list> <module>
```

ARGUMENTS

- Server URL
- Workspace Module

Server URL

serverURL	Specifies the URL of the module. Specify the URL as follows: sync://<host>[:<port>]/Modules/[<category...>/]<module> or syncs://<host>[:<port>]/Modules/[<category...>/]<module> where 'sync://' or 'syncs://' is required, <host> is the machine on which the SyncServer is installed, <port> is the SyncServer port number (defaults to 2647/2679), [<category...>} is the optional category (and/or sub-category) containing the module, and <module> is the name of the module. For example: sync://serv1.abco.com:1024/Modules/ChipDesigns/Chip

Workspace Module

<workspace module>	Specifies the workspace module instance for the module; for example: Chip%0.

OPTIONS

- -depth
- -dryrun
- -report
- -tags

- -xtras

-depth

-depth all|one|none	Indicates how many levels of the module hierarchy to send to the remote server hosting the associated Enterprise Design system. all - Synchronizes recursively through the entire hierarchy for each module version identified by each tag in the taglist specified with the -tags option. This option provides the most complete update to the server, but can be performance intensive. Hierarchical references to non-module objects are considered "leaf" objects and DesignSync does not attempt to continue traversal through that object or configuration. Hierarchical references to external modules are updated with the options specified with -xtras being passed, unaltered to the external CM system. one - Synchronizes the first level of the module hierarchy. For each module version identified by the tag list specified by the -tags option, follow the hierarchical references attached to that version, but do not traverse the hierarchy. This option minimizes the risk of an out-of-date hierarchy without the performance impact of updating the entire hierarchy. (Default) none - Synchronizes only the specified module version identified by the tag list specified by the -tags option; does not synchronize any hierarchical references. This may result in an incomplete hierarchy on the server.

-dryrun

-dryrun	Only reports the actions performed by the command, but does not actually perform any action. The command runs to completion, noting any errors, but continuing to run the command.

-report

-report brief|normal |verbose	Determines what information is returned in the output of the command. The command output is returned after the command has finished processing. Valid values are:

- brief - outputs the status of the running command, command results, and errors.

- normal - outputs the information contained in -brief mode and information about the enterprise versions being created. (Default),

- verbose - There is currently no difference between the verbose and normal reports.

Note: The report information is also passed to the ENOVIA server.

-tags

-tags <taglist> Specifies which tagged versions are processed by the command. The tags are specified in a comma-separated list. Tags may be either branch or version tags. Branch tags do not require a trailing semicolon. If a branch tag is specified, it is the branch itself that is specified as the work-in-progress revision.

You may use glob style regular expressions to specify a tag. All tags that match the specified tag expression are processed. For example, R* would match both of the following tags: REL01 and READY_FOR_TEST.

If there is no match for one or more glob style tag specified in the tag list, but all fully-specified tags exist, the command succeeds. If any fully-specified tag (ie: not containing a glob expression to be evaluated) does not exist, the entire command fails.

-xtras

-xtras <list> List of command line options to pass to the external module change management system. Any options specified with the -xtras option are sent verbatim, with no processing by the populate command, to the TCL script that defines the external module change management system.

Note: The external modules system is only accessed when the specified -depth option is all. If this option is specified with a different -depth level, it is silently ignored.

RETURN VALUE

```
This command does not return any TCL values.  If the command
succeeds, after command completion, DesignSync displays the values of
all set fields for the synchronized products. If the command fails,
it returns an error message explaining the failure.
```

SEE ALSO

```
command defaults, entobj setpolicy, entobj settype, entobj show,
populate, selectors, tag
```

entobj type

entobj type Command

NAME

```
entobj type          - Displays the product type for the module
```

DESCRIPTION

```
 This command displays the product type for the module.  If there is
 no product type set for the module, the command returns a null value
 ("").

This command is subject to access controls on the server.
```

SYNOPSIS

```
entobj type <module>
```

ARGUMENTS

- Module instance
- Server URL

Module instance

```
<Workspace Module>  The workspace module instance name; for example:
                    Chip%0.
```

Server URL

```
<Server URL>        The Server module URL in the format:
       sync[s]://<host>[:<port>]/Module/[<category>...]/<ModuleName>
                    where <host> is the SyncServer on which the module
                    resides, <port> is the SyncServer port number,
                    <category> identifies path to the module, and
                    <ModuleName> is the module associated with the
                    Enterprise Design object.
```

RETURN VALUE

```
This command does not return any TCL values. When successful, this
command returns the product type or, if no product type is assigned,
an empty string (""). If the command fails, DesignSync displays an
error message explaining the failure.
```

SEE ALSO

```
entobj policy, entobj setpolicy, entobj settype, entobj synchronize
,
```

EXAMPLES

- Example of Showing the Product Type

Example of Showing the Product Type

```
dss> entobj type sync://serv2.ABCo.com:2647/Modules/ChipDZ
type_Chip_ Design_DZ-2
```

Mcache Settings for Shared Developments

eda

eda Command

NAME

```
eda                 - Development module cache paths commands
```

DESCRIPTION

```
The eda commands allow you to manage your DesignSync development
module cache paths to share common modules, rather than duplicate the
information across multiple developments. These commands allow you to
add, remove and list additional module cache paths.
```

SYNOPSIS

```
eda <eda_command> [<eda_command_options>] <eda_command_arguments>

Usage:  [addmcachepath|createrefws|listmcachepath|removemcachepath]
```

ARGUMENTS

```
Varies by command.
```

OPTIONS

```
Varies by command.
```

RETURN VALUE

```
Varies by command.
```

SEE ALSO

```
eda addmcachepath, eda createrefws, eda listmcachepath, eda
removemcachepath, sda
```

eda addmcachepath

eda addmcachepath Command

NAME

```
eda addmcachepath   - Adds mcachepath to available server mcache paths
```

DESCRIPTION

```
The command adds the specified path to the set of additional module
cache paths for the server specified. The path is also added to the
list in the WSProjectRegistry.reg settings files for each development
managed by the server.

The mcache path is checked to see if
o The path exists
o The path specified is a modules root

New developments inherit any existing additional mcache paths.
```

```
This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

This command supports the command defaults system.
```

SYNOPSIS

```
eda addmcachepath -path <path>[,<path>...] [-[no]replace]
                  [-[no]validate] <ServerURL>
```

ARGUMENTS

- Server URL

Server URL

```
<ServerURL>     Specifies the URL of the development server hosting
                the newly added mcache paths. Specify the URL as
               follows:
                sync://<host>[:<port>] or
                syncs://<host>[:<port>] where 'sync://' or 'syncs://'
                is required, <host> is the machine on which the
                SyncServer is installed, and <port> is the SyncServer
                port number (defaults to 2647/2679).
                For example:  sync://serv1.abco.com:1024
```

OPTIONS

- -path
- -[no]replace
- -[no]validate

-path

```
-path <path>    Absolute or relative path to the mcache path.
 [,<path>...]   You can specify multiple paths in a list of
                space-separated path names.  To specify multiple
                paths, surround the path list with double
                quotation marks ("") and separate path names
                with a comma.
               For example:"/dir/cacheA,/dir2/cacheB".

                The path list can contain both module and legacy
                module mcache paths. For a module cache the path to
                the root directory of the module cache must be
                supplied.
                Note: To specify a path that includes spaces:
                      o In stcl or stclc, surround the path
                        containing the spaces with curly braces.
```

```
                    For example:
                    "/dir1/cache {/dir2/path name}"
                  o  In dss or dssc, use backslashes (\) to
                     'escape' the spaces. For example:
                     "/dir1/cache /dir2/path\ with\ spaces"
```

-[no]replace

```
 -[no]replace    Determines whether to replace the mcache path
                 information with the new mcachepaths specified by the
                 -path option.

                 -replace replaces the list of mcache paths for the
                  server with the new list specified for the -path
                  option.

                 -noreplace adds the paths specified with -paths and
                  does not remove any existing paths. (Default)
```

-[no]validate

```
 -[no]validate   Determines whether to validate the accessibility of
                 the mcache path specified on the server.

                -validate checks whether the path specified by the
                 -path option exists and is a modules root.

                 -novalidate does not do any verification of paths
                  specified with the -path option. (Default)
```

RETURN VALUE

```
On command success, returns the list of defined mcache paths. On
failure, displays an error message explaining the failure.
```

SEE ALSO

```
command defaults, eda listmcachepath, eda removemcachepath
```

EXAMPLES

- Example Showing Adding Paths to the Mcache Path List
- Example Showing Replacing the Paths in the Mcache Path List

Example Showing Adding Paths to the Mcache Path List

```
This example shows adding a path to the mcachepath list.  Note that
in the output, you see the full collection of paths, separated by a
space. The path must be a module root.

dss> eda addmcachepath -path /home/rsmith/workspaces/ -validate
 sync://serv1.ABCo.com:2647/Modules/ChipNZ-47

/DesignSync/mcachestore /home/rsmith/workspaces
```

Example Showing Replacing the Paths in the Mcache Path List

```
This example shows replacing the paths in the mcache path list with
the path(s) specified.

dss> eda addmcachepath -path "/home/mcachestore1,/home/mcachestore2"
 -validate -replace sync://serv1.ABCo.com:2647/ModulesChipNZ-47

/home/mcachestore1 /homemcachestore2
```

eda createrefws

eda createrefws Command

NAME

```
eda createrefws     - Create reference workspace for development
```

DESCRIPTION

```
The eda createrefws command creates a reference workspace used by sda
mk for enhanced performance when creating a new development area. The
reference workspace is created as a mirror so is automatically
maintained for the most current and accurate workspace state.

Before creating the reference workspace, DesignSync verifies that:
    o The reference workspace directory does not already exist
      parallel to the Data Replication Root (DRR) for the development
      instance. If it already exists, and is marked as a reference
      workspace, the operation returns success, but does not make any
      changes.

   o If a reference workspace exists, it must either be the workspace
     root directory or can be set as the module root directory.

   o File cache reference counting is disabled so the reference
     workspace is created without cache reference counting.

   o Partition based file caching must be disabled so soft links,
```

```
     rather than hard links are created to the file cache.

This command supports the access control system.
```

SYNOPSIS

```
eda createrefws [-assignment <AssignmentName>] -name <DevName> <ServerUrl>
```

ARGUMENTS

- Server URL

Server URL

```
<ServerURL>     Specifies the URL of the development server Specify
                the URL as follows:
                sync://<host>[:<port>] or
                syncs://<host>[:<port>] where 'sync://' or 'syncs://'
                is required, <host> is the machine on which the
                SyncServer is installed, and <port> is the SyncServer
                port number (defaults to 2647/2679).
                For example:  sync://serv1.abco.com:1024
```

OPTIONS

- -assignment
- -name

-assignment

```
[-assignment       The name of the assignment used for the reference
 <AssignmentName>] workspace mirror.  Specify the same assignment as the one
                   associated with the development area.

                  If the assignment is not specified, DesignSync
                   uses the first assignment found.

                   If the selector for the assignment is "Default,"
                   then the developmentâ€™s selector is used for the
                   mirror. If no selector is provided in the assignment
                   or the development instance, DesignSync uses the
                   default "Trunk" selector.
```

-name

```
-name <DevName>    Name of the development. Development names are
```

```
                    case sensitive.
```

RETURN VALUE

```
 The command does not return a Tcl value.  If the command succeeds,
 DesignSync creates the reference workspace and displays a success
 message containing some of the reference workspace properties. If the
 command fails, DesignSync displays a message to explain the failure.
```

SEE ALSO

```
 duplicatews, mirror create, sda mk
,
```

eda listmcachepath

eda listmcachepath Command

NAME

```
 eda listmcachepath  - Lists mcachepath to available server mcache paths
```

DESCRIPTION

```
 This command returns a list of the additional mcachepaths defined for
 the specified server. If no mcachepaths are defined, the command
 returns an empty list.

 This command is subject to access controls on the server. See the
 ENOVIA Synchronicity Access Control Guide for details.

 This command supports the command defaults system.
```

SYNOPSIS

```
 eda listmcachepath  [-format list|text] <ServerURL>
```

ARGUMENTS

- Server URL

Server URL

```
<ServerURL>     Specifies the URL of the development server hosting
                the mcache directories. Specify the URL as
               follows:
                sync://<host>[:<port>] or
                syncs://<host>[:<port>] where 'sync://' or 'syncs://'
                is required, <host> is the machine on which the
                SyncServer is installed, and <port> is the SyncServer
                port number (defaults to 2647/2679).
                For example:  sync://serv1.abco.com:1024
```

OPTIONS

- -format

-format

```
-format         Specifies the way the output is returned.
                The default is text. The format 'text' will return
                each path on a new line in the format name=value. The
               format 'list' will list the values in a Tcl list in
               the form {name1 value1 name2 value2 ...}
```

RETURN VALUE

```
When the command runs successfully in -format list mode, returns a
Tcl list of mcachepaths defined for the specified server. When the
command runs successfully in -format text mode, returns a list of
paths. When the command fails, returns an error explaining the
failure.
```

SEE ALSO

```
command defaults, eda addmcachepath, eda removemcachepath
```

EXAMPLES

- Example Showing A List of the Mcache Paths in Text Format
- Example Showing A List of the Mcache Paths in TCL List Format

Example Showing A List of the Mcache Paths in Text Format

```
This example shows a listing of the additional mcache paths formatted
for easy reading.

dss> eda listmcachepath sync://serv1.ABCo.com:2647/Modules/ChipNZ-47
/DesignSync/mcachestore
/home/rsmith/workspaces
C:/My Mirrors/workspaces
```

Example Showing A List of the Mcache Paths in TCL List Format

```
This example shows a listing of the additional mcache paths formatted
for TCL processing.
dss> eda listmcachepath -format list \
sync://serv1.ABCo.com:2647/Modules/ChipNZ-47

 /DesignSync/mcachestore /home/rsmith/workspaces {C:/My Mirrors/workspaces}
```

eda removemcachepath

eda removemcachepath Command

NAME

```
eda removemcachepath - Removes mcachepaths from the specified server
```

DESCRIPTION

```
This command removes a path from the set of additional module
cache paths for the development server specified.  The path is also
removed from the list in the registry setting in
WSProjectRegistry.reg for each development managed by the server.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.
```

SYNOPSIS

```
eda removemcachepath -path <path> <ServerURL>
```

ARGUMENTS

- Server URL

Server URL

```
<ServerURL>     Specifies the URL of the development server hosting
                the mcache paths being removed. Specify the URL as
               follows:
                sync://<host>[:<port>] or
                syncs://<host>[:<port>] where 'sync://' or 'syncs://'
                is required, <host> is the machine on which the
                SyncServer is installed, and <port> is the SyncServer
                port number (defaults to 2647/2679).
```

```
                For example:  sync://serv1.abco.com:1024
```

OPTIONS

- -path

-path

```
-path <path>    Absolute or relative path to the mcache path.
                If the relative path is specified, the path is
               evaluated to create the absolute path before being
               removed.


                You can only remove one path.
```

RETURN VALUE

```
Returns 0 on success.  On failure, displays an error message
explaining the failure.
```

SEE ALSO

```
eda addmcachepath, eda listmcachepath
```

EXAMPLES

- Example of Removing a Path from the Mcache Path list

Example of Removing a Path from the Mcache Path list

```
This example shows removing a path from the additional list of mcache
paths.
dss> eda removemcachepath -path "C:/My Mirrors/workspaces" \
sync://serv1.ABCo.com:2647/Modules/ChipNZ-47
 /DesignSync/mcachestore /home/rsmith/workspaces
```

URL Sync Object Model

url Commands

NAME

```
url                   - URL navigation commands
```

DESCRIPTION

```
These commands return a value, enabling the user to access the
Synchronicity Object Model (SOM) of information. This includes
going from folders to files, from files to their vaults, from
vaults to the versions inside them, and so on. All commands are
proceeded by the super command "url".

Note: url commands provide information about files and folders
      in your DesignSync work areas. Do not use these commands
      to obtain information about local mirror directories.
      You can use these commands to obtain information about all
      standard mirror directories.

Most url commands accept either relative or absolute URL paths.
For example, both of the following are valid:

   stcl> url vault .           # relative
   stcl> url vault [spwd]      # absolute

The following commands require an absolute path:
   url projects, url users

Note: The url commands are available from all DesignSync client
      shells. The stclc/tcl shells allow you to operate on the
      values returned by the url commands but the dss/dssc shells
      do not. Thus these commands are more useful in stcl/tcl
      than in dss/dssc.
```

SYNOPSIS

```
url <url_command> [<url_command_options>] <object>

Usage: url  [branchid|configs|container|contents|exist|fetchedstate|
            fetchtime|filter|getprop|inconflict|leaf|locktime|members|
            mirror|modified|naturalpath|notes|owner|path|projects|
            properties|registered|relations|resolveancestor|
            resolvetag|retired|root|selector|servers|setprop|syslock|
```

```
            tags|users|vault|versionid|versions|view]
```

OPTIONS

```
Varies by command.
```

RETURN VALUE

```
Varies by command.
```

SEE ALSO

```
stcl
```

EXAMPLES

```
See specific url commands.
```

url

url Commands

NAME

```
url                  - URL navigation commands
```

DESCRIPTION

```
These commands return a value, enabling the user to access the
Synchronicity Object Model (SOM) of information. This includes
going from folders to files, from files to their vaults, from
vaults to the versions inside them, and so on. All commands are
proceeded by the super command "url".

Note: url commands provide information about files and folders
      in your DesignSync work areas. Do not use these commands
      to obtain information about local mirror directories.
      You can use these commands to obtain information about all
      standard mirror directories.
```

```
Most url commands accept either relative or absolute URL paths.
For example, both of the following are valid:

   stcl> url vault .           # relative
   stcl> url vault [spwd]      # absolute

The following commands require an absolute path:
   url projects, url users

Note: The url commands are available from all DesignSync client
      shells. The stclc/tcl shells allow you to operate on the
      values returned by the url commands but the dss/dssc shells
      do not. Thus these commands are more useful in stcl/tcl
      than in dss/dssc.
```

SYNOPSIS

```
url <url_command> [<url_command_options>] <object>

Usage: url  [branchid|configs|container|contents|exist|fetchedstate|
            fetchtime|filter|getprop|inconflict|leaf|locktime|members|
            mirror|modified|naturalpath|notes|owner|path|projects|
            properties|registered|relations|resolveancestor|
            resolvetag|retired|root|selector|servers|setprop|syslock|
            tags|users|vault|versionid|versions|view]
```

OPTIONS

```
Varies by command.
```

RETURN VALUE

```
Varies by command.
```

SEE ALSO

```
stcl
```

EXAMPLES

```
See specific url commands.
```

url branchid

url branchid Command

NAME

```
url branchid        - Returns the branch number of an object.
```

DESCRIPTION

```
This command returns the branch number of the specified object.

o  For managed objects, the url branchid command returns the current
   branch number (as stored in the local metadata).
o  For branch and version objects, the url branch id command returns
   the branch number.
o  For vaults and for versionable objects that are not under revision
   control, the url branch id command returns "1", (because 1 is the
   default branch number).
o  For all other object types, or if the specified object does not
   exist, an exception is thrown.
```

SYNOPSIS

```
url branchid [--] <argument>
```

ARGUMENTS

- Module Member (Module-based)
- Workspace Module (Module-based)
- DesignSync Object (File-based)

```
Specifies one of the following arguments:
```

Module Member (Module-based)

```
<module member>     Specifies the module member for which you want
                    the branch id of the module it belongs to.
                    You can also specify the module member branch
                    or the module member version as arguments.
```

Workspace Module (Module-based)

```
<workspace module>  Specifies the workspace module for which you want
                    the current branch is.

                    Note: The server module is not a valid argument
                          for this command.
```

DesignSync Object (File-based)

```
<DesignSync object> Specifies the DesignSync object for which you
                    want the current branch number as stored in the
                    metadata.
```

OPTIONS

- --

--

```
--                  Indicates that the command should stop looking
                    for command options. Use this option when the
                    argument to the command begins with a hyphen
                    (-).
```

RETURN VALUE

```
For valid objects, returns the branch number of the specified
object. For objects not under revision control, returns 1 (which is
identifer for the default branch, "Trunk:".

For invalid or non-existant objects, returns an error.
```

SEE ALSO

```
url versionid
```

EXAMPLES

- Examples of Displaying Branch ID (Module-based)
- Examples of Displaying the Branch ID (File-based)

```
The following examples show the different return values for
```

```
the url branchid command.

For unmanaged objects and vaults, returns "1":
  stcl> url branchid test.s19
  1
  stcl> url branchid [url vault samp.asm]
  1

For invalid arguments or if the object does not exist,
returns the following error messages.
  stcl> url branchid .
  SomAPI-E-101: The specified object is not associated with a branch
  or version.
  stcl> url branchid [url vault .]
  SomAPI-E-101: The specified object is not associated with a branch
  or version.
  stcl> url branchid nosuchobject
  SomAPI-E-101: Object does not exist at specified URL.
```

Examples of Displaying Branch ID (Module-based)

```
For modules, returns the branch id of the module in your workspace
   stcl> url branchid Mod1
   1.3.1

For module members, returns the branch id of the module it belongs
to.
  stcl> url branchid File1.txt
  1.1.1
```

Examples of Displaying the Branch ID (File-based)

```
  stcl> ls -report RH samp.asm samp.lst test.mem test.s19
  Version          Branch Tags  Name
  -------          -----------  ----
  1.2                    Trunk  samp.asm
  1.1 -> 1.2             Trunk  samp.lst
  1.2.1.2               rel1.2  test.mem
  Unmanaged                     test.s19

For local managed objects, returns the current branch number:
  stcl> url branchid samp.asm
  1
  stcl> url branchid samp.lst
  1
  stcl> url branchid test.mem
  1.2.1

For versions and branches, returns the branch number:
  stcl> url branchid [url vault test.mem]1.2.1
  1.2.1
  stcl> url branchid [url vault test.mem]1.2.1.2
```

1.2.1

url configs

url configs Command

NAME

```
url configs          - Returns the configurations of a ProjectSync project
```

DESCRIPTION

```
This command returns the list of configurations for a
ProjectSync-defined project. The project and associated
configurations must be defined from ProjectSync -- see the
ProjectSync User's Guide for more information. You can use this
command in conjunction with 'url contents' to see the
DesignSync-tagged versions that are associated with a
ProjectSync-defined configuration.

The specified object can be a project's vault folder or a
corresponding local working folder. Any other object type
results in an empty list.
```

SYNOPSIS

```
url configs [--] <argument>
```

ARGUMENTS

- Workspace Folder
- Server Folder

```
Specifies one or more of the following arguments:
```

Workspace Folder

```
<DesignSync folder> Specifies the workspace folder of a project for
                    which you want a list of configurations.
```

Server Folder

```
<server folder>     Specifies the vault folder of a project for
                    which you want a list of configurations.
```

OPTIONS

- --

--

```
--              Indicates that the command should stop looking for
                command options. Use this option when the argument
                to the command begins with a hyphen (-).
```

RETURN VALUE

```
For valid objects, returns list of configurations.

For other objects: Returns an empty list.
```

SEE ALSO

```
url contents
```

EXAMPLES

```
This example returns the ProjectSync-defined configurations for the
Sportster project, first by specifying the project folder on the
SyncServer, then the corresponding local folder. Sportster has two
configurations: Alpha and Gold.

  dss> url configs sync://holzt:2647/Projects/Sportster
  sync://holzt:2647/Projects/Sportster@Alpha
  sync://holzt:2647/Projects/Sportster@Gold
  dss> url configs /home/goss/Projects/Sportster
  sync://holzt:2647/Projects/Sportster@Alpha
  sync://holzt:2647/Projects/Sportster@Gold

You can then use 'url contents' to see what DesignSync-tagged
versions correspond to a project configuration. Note that 'url
contents' is not recursive -- see the help for 'url contents' for
details. In this example, the Alpha configuration has one subfolder
(code) associated with the configuration, which contains two versions
```

```
that were tagged with the DesignSync tag associated with the
ProjectSync Alpha configuration.

  dss> url contents sync://holzt:2647/Projects/Sportster@Alpha
  sync://holzt:2647/Projects/Sportster/code@Alpha
  dss> url contents sync://holzt:2647/Projects/Sportster/code@Alpha
  sync://holzt:2647/Projects/Sportster/code/samp.lst;1.1
  sync://holzt:2647/Projects/Sportster/code/samp.mem;1.1
```

url container

url container Command

NAME

```
url container       - Returns the object containing a specified
                      object
```

DESCRIPTION

```
This command returns the URL of the object (such as a folder)
containing the specified object (such as a file).

Note: In stcl/stclc mode, when specifying a URL that contains a
semicolon (;), surround the URL with double quotes.
```

SYNOPSIS

```
url container [--] <object>
```

OPTIONS

- --

--

```
--              Indicates that the command should stop looking for
                command options. Use this option when the argument
                to the command begins with a hyphen (-).
```

RETURN VALUE

```
For a client-side versionable objects, returns client-side
folder. For server-side objects, returns the vault folder. For
projects, returns the vault folder containing the project. For
server-side note types, returns parent URL. For server-side notes,
returns the note type URL.

For other objects: Returns parent folder, or if no parent folder,
returns an empty list.  Note: 'url container' does not verify that
the object exists.
```

SEE ALSO

```
url contents
```

EXAMPLES

- Example Returning the Local Folder that Contains the Object
- Example Returning the Server Folder that Contains the Object

Example Returning the Local Folder that Contains the Object

```
This example returns the folder that contains top.v.
  dss> url container top.v
  file:///home/Projects/Sportster/synth
```

Example Returning the Server Folder that Contains the Object

```
This example returns the server folder that contains the top.v
vault.
  stcl> url container [url vault top.v]
  sync://server.company.com:port/Projects/Sportster/synth
```

url contents

url contents Command

NAME

```
url contents        - Returns the objects in a container object
```

DESCRIPTION

- Notes for Modules (Module-based)

This command returns a list of URLs of the objects contained in the specified container object, such as a folder or configuration. If the object is not appropriate for the requested operation, an empty list is returned.

The 'url contents' command is not recursive. For example, 'url contents' on a ProjectSync configuration always returns folders as part of the configuration. You can then invoke 'url contents' on each subfolder in the project.

Note: In stcl/stclc mode, when specifying a URL that contains a semicolon (;), surround the URL with double quotes.

Notes for Modules (Module-based)

The "url content" command shows the contents of a module folder, but is not used for examining the contents of a module and therefore does not accept module as a valid argument type. Use "ls" and "contents" commands to list the full module contents.

SYNOPSIS

```
url contents [-all | -ifpopulated [-incremental]] [-prefetch]
             [-version <selector>[,<selector>...]] [--] <argument>
```

ARGUMENTS

- Module Folder (Module-based)
- DesignSync Folder (File-based)
- DesignSync Vault (File-based)

Specify one or more of the following arguments:

Module Folder (Module-based)

<module folder>	Returns a list of the URLs of files and folders contained in the specified workspace module folder.

DesignSync Folder (File-based)

<DesignSync folder>	Returns a list of URLs of files and folders contained in the specified folder in the

```
                    workspace.
```

DesignSync Vault (File-based)

```
 <DesignSync vault>   Returns a list of the URLs of the different
                      vault versions checked into the specified vault.
```

OPTIONS

- -all
- -ifpopulated
- -incremental
- -prefetch
- -version
- --

-all

```
 -all                 Reports the objects in the local folder as well
                      as those objects that would be there if it were
                     fully populated with the contents of the
                     associated vault. If the object is a vault-side
                     object (a vault, version, or branch), this
                     option is ignored.

                      This option is mutually exclusive with
                      -ifpopulate.
```

-ifpopulated

```
 -ifpopulated         Report the contents of the local folder if it
                      were fully populated with the contents of its
                      associated vault.

                      You can also specify -incremental to limit the
                      result to return only those objects that would
                      return in an incremental populate as opposed to
                      a full populate.  The list is empty if the
                      object is not a local folder.

                      This option is mutually exclusive with -all.
```

-incremental

```
 -incremental         Modifies -ifpopulated to limit the result to
```

	return the URLs of only those objects that would return in an -incremental populate. Note: You must have at some time performed a full populate on the folder for the -incremental option to work properly.

-prefetch

-prefetch	Used for advanced programming; exposes an optimization to the caller. If used, the call to contents is slower, but the subsequent enumeration of the returned list has extra information cached. If the caller needs to enumerate the contents and for each object call commands such as 'url tags' or 'url properties', overall performance is better if this option is used. If the caller needs to retrieve only the names of the objects, this option makes the operation slower.

-version

-version <selector>	Use with -ifpopulate or -all. Specifies the selector list (typically branch or version tag) to use for the hypothetical populate. The default (-version not specified) is to inherit the selector from the parent folder. Note: To use -version to specify a branch, specify both the branch and version as follows: '<branchtag>:<versiontag>', for example, 'Rel2:Latest'. You can also use the shortcut, '<branchtag>:', for example "Rel2:". If you don't explicitly specify the branch selector in this way, DesignSync does not resolve the selector as a branch selector.

--

--	Indicates that the command should stop looking for command options. Use this option when the argument to the command begins with a hyphen (-).

RETURN VALUE

```
For a client-side folder (Asic): Returns list of client-side folders
and files (file://home/karen/Asic/Sub file://home/karen/Asic/x.v).

For a vault ("sync://holzt:2647/Projects/Asic/x.v;"): Returns vault
versions: ({sync://holtz:2647/Projects/Asic/x.v;1.1}
{sync://holtz:2647/Projects/Asic/x.v;1.2}
{sync://holtz:2647/Projects/Asic/x.v;1.3}).

For a version ("sync://holzt:2647/Projects/Asic/x.v;1.1",
"sync://holzt:2647/Projects/Asic/x.v;yellow"): Not a
container object; returns an empty list.

For branch ("sync://holzt:2647/Projects/Asic/x.v;1.5.1",
"sync://holzt:2647/Projects/Asic/x.v;Golden:Latest"): Not a
container object; returns an empty list.

For a project (sync://holzt:2647/Projects/Asic): Returns vault
folder containing project: (sync://holzt:2647/Projects).

For a configuration (sync://holzt:2647/Projects/Asic/Sub@Rel1):
Returns vault folder: (sync://holzt:2647/Projects/Asic).

For a server-side note system URL (sync:///Note)
Returns the list of note systems on the server (currently the
only note system is SyncNotes)

For a server-side SyncNotes URL (sync:///Note/SyncNotes):
Returns the list of URLs for all note types defined on the server.

For a server-side note-type URL (sync:///Note/SyncNotes/HW-Defect-1):
Returns a list of URLs of all notes of type HW-Defect-1.

For a server-side Users URL (sync:///Users)
Returns a list of URLs for all user profiles on the server.

For a module folder, returns a list of members in that folder.

For other objects: Returns an empty list.
```

SEE ALSO

```
note systems, notetype enumerate, populate, selectors, url container,
url notes, url users, url versions
```

EXAMPLES

- Example Showing the Contents of a Module Folder (Module-based)
- Sample File Structure for Examples (File-based)
- Example of Local Folder Contents (File-based)
- Example of Vault Folder Contents (File-based)
- Example of Returning the Contents of a Branch (File-based)

Example Showing the Contents of a Module Folder (Module-based)

```
 In the following example, the workspace //MyModules contains the
 following:
   MyModules
     File1.txt
     File2.txt
     File3.txt
     File4.txt
 Return contents of local folder MyModules

   stcl> url contents /home/tachatterjee/MyModules
file:///home/tachatterjee/MyModules/File1.txt
file:///home/tachatterjee/MyModules/File2.txt
file:///home/tachatterjee/MyModules/File3.txt
file:///home/tachatterjee/MyModules/File4.txt
```

Sample File Structure for Examples (File-based)

```
 In the following examples, the local work area
 /Projects/ASIC contains the following:
   ASIC
     FileA          # Three versions exist in the vault
     FileB          # There is a new version in the vault
     FileC          # Not under revision control
     FileD          # In the vault, but not in local working directory
     Decoder
       FileE        # There is a new version in the vault
       FileF        # Only file to have tag 'Gold' (version 1.2)
```

Example of Local Folder Contents (File-based)

```
 Return contents of local folder ASIC

   dss> url contents /Projects/ASIC
   file:///Projects/ASIC/Decoder
   file:///Projects/ASIC/FileA
   file:///Projects/ASIC/FileB
   file:///Projects/ASIC/FileC
```

Example of Vault Folder Contents (File-based)

```
Return contents of vault folder ASIC

  dss> url contents sync://holzt:2647/Projects/ASIC
  sync://holzt:2647/Projects/ASIC/Decoder
  sync://holzt:2647/Projects/ASIC/FileA;
  sync://holzt:2647/Projects/ASIC/FileB;
  sync://holzt:2647/Projects/ASIC/FileD;
```

Example of Returning the Contents of a Branch (File-based)

```
Return contents of FileA main branch

  dss> url contents "sync://holzt:2647/Projects/ASIC/FileA;1"
  sync://holzt:2647/Projects/ASIC/FileA;1.1
  sync://holzt:2647/Projects/ASIC/FileA;1.2
  sync://holzt:2647/Projects/ASIC/FileA;1.3
```

Example Showing the Contents Resulting from Full Populate (File-based)

```
Return contents of /Project/ASIC resulting from full populate

  dss> url contents -ifpopulated /Projects/ASIC
  file:///Projects/ASIC/Decoder
  file:///Projects/ASIC/FileA
  file:///Projects/ASIC/FileB
  file:///Projects/ASIC/FileD
```

Example Showing the Contents Resulting From Incremental Populate (File-based)

```
 Return updated objects of /Project/ASIC after incremental populate

  dss> url contents -ifpopulated -incremental /Projects/ASIC
  file:///Projects/ASIC/Decoder
  file:///Projects/ASIC/FileB
  file:///Projects/ASIC/FileD
```

Example Showing Contents Resulting From Populate with Selector (File-based)

```
Return what is populated when the selector is "Gold", which can be a
version or branch tag

  dss> url contents -ifpopulated -version Gold /Projects/ASIC
  file:///Projects/ASIC/Decoder
```

Example Showing Contents Resulting from Populate with Configuration (File-based)

```
Return the contents of the ASIC project's "Gold" configuration

  dss> url contents sync://holzt:2647/Projects/ASIC@Gold
  sync://holzt:2647/Projects/ASIC/Decoder@Gold

  dss> url contents sync://holzt:2647/Projects/ASIC/Decoder@Gold
  sync://holzt:2647/Projects/ASIC/Decoder/FileF;1.2
```

url exists

url exists Command

NAME

```
url exists          - Reports whether an object exists
```

DESCRIPTION

```
This command determines whether an object physically exists either in
the workspace or in the vault. If it exists, returns 1, if not
returns 0.
```

SYNOPSIS

```
url exists [--]  <argument>
```

ARGUMENTS

- Module (Module-based)
- Module Member (Module-based)
- Module Folder (Module-based)
- DesignSync Object (File-based)

```
Specify one of the following arguments:
```

Module (Module-based)

```
<module>            Specifies the module whose existence you want to
                    verify.
                    Returns 1 if the module, module branch or the
                    module version exists. Else returns 0. If a
                    workspace module is specified, it returns whether
```

```
                    that exists. Otherwise returns whether the server
                    module, branch or version exists.
```

Module Member (Module-based)

```
<module member>     Specifies the module member whose existence you want
                    to verify.
                    Returns 1 if the workspace module member
                    exists. Else returns 0.
                    Note: The "url exists" command is not applicable
                    for server module member, branch or version. It is
                    not meaningful to request whether a member exists
                    on the server, as a member exists only in the
                    context of a module version.
                    You can run the "url vault" command on a member to
                    get the sea-of-vaults object, then run the "url
                    exists" command on that object. However, the "url
                    vault" command succeeds if the object is checked
                    in, and the "url exists" command fails only if a
                    catastrophic event causes the member vault to be
                    deleted.
```

Module Folder (Module-based)

```
<module folder>     Specifies the workspace module folder whose existence
                    you want to verify.
                    Returns 1 if the workspace module folder
                    exists. Else returns 0.
                    Note: The "url exists" command is not applicable
                    for server module folder. If a server folder under
                    the module server path is specified, it returns 0
                    as an unsupported argument type.
```

DesignSync Object (File-based)

```
<DesignSync object> Specifies the DesignSync object whose existence
                    you want to verify.
                    Returns 1 if it physically exists.  Else returns
                    0. Returns the same value for vault, branch,
                    version, folder, note, project and unmanaged objects.

                    Note: A DesignSync reference to a checked-in file
                    returns the value 1 to distinguish it from a file
                    that was never brought into the user's work area.
```

OPTIONS

- --

--

```
--              Indicates that the command should stop looking for
                command options. Use this option when the argument
                to the command begins with a hyphen (-).
```

RETURN VALUE

```
For all existing objects: Returns 1 (Tcl TRUE).
For nonexistent objects: Returns 0 (Tcl FALSE).
```

SEE ALSO

```
url modified, url registered, url retired
```

EXAMPLES

- Example of Verifying the Existence of a Module (Module-based)
- Example of Verifying the Existence of a Module Member (Module-based)
- Example of Verifying the Existence of File-Based Objects (File-based)

Example of Verifying the Existence of a Module (Module-based)

```
This example uses 'url exists' to verify the existence of a
module:
  stcl> url exists sync://srv2.ABCo.com:2647/Modules/JitaMod1
  1
```

Example of Verifying the Existence of a Module Member (Module-based)

```
This example uses 'url exists' to verify the existence
of a module member:
  stcl> url exists File1.txt
  1
```

Example of Verifying the Existence of File-Based Objects (File-based)

```
This example uses 'url exists' to verify the existence of several
files.
```

```
stcl> ls -report OR top*
Object Type      Version          Name
-----------      -------          ----
File             1.2              top.gv
File             1.2 -> 1.3       top.v
File             Unmanaged        top.log
Referenced File  Refers to: 1.1   top.f

stcl> url exists top.gv
1
stcl> url exists top.f
1
stcl> url exists top.log
1
stcl> url exists top.txt
0
```

url fetchedstate

url fetchedstate Command

NAME

```
url fetchedstate    - Returns the fetched state of an object
```

DESCRIPTION

```
This command returns the fetched state of the specified
object. The fetch state answers the question: "How was this object
checked out into my local work area?" Possible states are:

 Lock       - Object was checked out with a lock. Note that the
              object is not necessarily still locked; another
              user could have unlocked it.
 Copy       - Object was checked out unlocked (replica).
 Mirror     - Object was checked out as a link to an object in
              the mirror directory.
 Cache      - Object was checked out as a link to an object in
              the cache.
 Reference  - Object was checked out as a reference.
              Note: For locked references, the fetched state
              returned is 'Lock'.
 NotFetched - Object was not fetched using DesignSync. The
              object is one of the following:
                o Not versionable (folder, version, and so on)
                o Not under revision control
                o Under revision control, but not fetched into
                  the work area by DesignSync (for example,
                  could be a tool's output, or could have been copied
                  at the operating-system level)
```

```
Note: If the object is not under revision control and is a link, a
      return value of "Mirror" instead of "NotFetched" can result.
```

SYNOPSIS

```
url fetchedstate [--] <argument>
```

ARGUMENTS

- Module Member (Module-based)
- DesignSync Object (File-based)

```
Specifies one of the following arguments:
```

Module Member (Module-based)

```
<module member>     Specifies the module member for which you want
                    the fetched state.

                    Note: The -modulecontext option is not required
                          here as this command can only be call upon
                          individual members in the workspace.
```

DesignSync Object (File-based)

```
<DesignSync object> Specifies the DesignSync object for which you want
                    the fetched state.
```

OPTIONS

- --

--

```
--                  Indicates that the command should stop looking for
                    command options. Use this option when the argument
                    to the command begins with a hyphen (-).
```

RETURN VALUE

```
Returns one of the following strings "Lock", "Copy", "Mirror",
"Cache", "Reference", "NotFetched" as indicated below:

For a managed object (specify as client-side object: (Asic/x.v or
file://.../Asic/x.v): Returns "Lock", "Copy", "Mirror", "Cache",
or "Reference" depending on the command used to fetch the object.

For an object not under revision control: Returns "NotFetched".

For a link not under revision control: Returns "Mirror".

For a nonversionable object (not a folder, collection,  nor file):
Returns "NotFetched".

For an object under revision control, but not fetched into the
work area by DesignSync: Returns "NotFetched".
```

SEE ALSO

```
ls, url registered
```

EXAMPLES

- Example Showing Fetch State of a Module (Module-based)
- Example Showing Fetch State of a Module Member (Module-based)
- Example Showing Fetch State of File-Based Objects (File-based)

Example Showing Fetch State of a Module (Module-based)

```
This example uses 'url fetchedstate' to return the fetched states
for modules:

  stcl> url fetchedstate /home/tachatterjee/JitaMOD
  NotFetched
```

Example Showing Fetch State of a Module Member (Module-based)

```
This example uses 'url fetchedstate' to return the fetched states
for module members:
  stcl>url fetchedstate File1.txt
  Copy
```

Example Showing Fetch State of File-Based Objects (File-based)

```
This example uses 'url fetchedstate' to return the fetched states
for several objects:

  dss> ls -report OR top*
  Object Type      Version          Name
  -----------      -------          ----
  File             1.2              top.gv
  File             1.2 -> 1.3       top.v
  File             Unmanaged        top.log
  Referenced File  Refers to: 1.1   top.f

  dss> url fetchedstate top.gv
  Copy
  dss> url fetchedstate top.v
  Lock
  dss> url fetchedstate top.log
  NotFetched
  dss> url fetchedstate top.f
  Reference
```

url fetchtime

url fetchtime Command

NAME

```
url fetchtime       - Returns the time when an object was fetched
```

DESCRIPTION

```
This command returns when the specified object was checked out
into your work area. The 'url fetchtime' command actually returns the
timestamp of the object when the object was fetched, not the
time of the fetch itself. Therefore, the value returned by
'url fetchtime' depends on whether or not you specified the -retain
option when you fetched the object.

Note: If you used 'populate -mirror' to fetch the object to your work
area, then a 'url fetchtime' operation for the object always returns 0.

The fetch time is unaffected by making local modifications to the
object. In fact, the fetch time and modification time being
different is an indicator that the file has been locally modified.

Specify an object in your work area as the argument to 'url
fetchtime'.
```

SYNOPSIS

```
url fetchtime [--] <argument>
```

ARGUMENTS

- Module (Module-based)
- Module Member (Module-based)
- DesignSync Object (File-based)

```
Specify one of the following arguments:
```

Module (Module-based)

```
<module>            Specifies the module for which you want the timestamp
                    when it was fetched into your work area.

                    For workspace modules, returns 0.
```

Module Member (Module-based)

```
<module member>     Specifies the member for which you want the time
                    when it was last fetched into your work area.
                    Note: A module context is not required as this
                    command can only be run on individual items
                    present in the workspace.
```

DesignSync Object (File-based)

```
<DesignSync object> Specifies the DesignSync object for which you want
                    the time when it was fetched into your
                    work area.
```

OPTIONS

- --

--

```
  --                Indicates that the command should stop looking for
                    command options. Use this option when the object
                    you specify begins with a hyphen (-).
```

RETURN VALUE

```
For a managed object specified as a client-side object
(Asic/x.v or file://.../Asic/x.v): Returns the fetched time in
time_t format, which is the number of seconds since the birth
of UNIX -- January 1, 00:00:00, 1970 (GMT).

For an object reference (or other objects that
lack fetch time metadata): Returns 0.

For an object not under revision control: Returns 0.

For a nonversionable object (not a folder, collection, nor file):
Returns 0.

Note: You can use the Tcl 'clock format' command to convert the
time_t format to a date string.
```

SEE ALSO

```
url fetchedstate, url locktime, url properties
```

EXAMPLES

Example Showing the Last Fetchtime of a Module (Module-based)

```
This example uses 'url fetchtime' to get the time when the module
Module1 was last fetched to the workarea:

stcl> url fetchtime Module1
0

Note: Since the workspace module was used as the argument, the result
is 0.
```

Example Showing Last Fetchtime of a Module Member (Module-based)

```
This example uses 'url fetchtime' to get the time when the module
member File1.txt was last fetched to the workarea:
  stcl> url fetchtime File1.txt
  1163014379
```

Example Showing Last Fetchtime of a DesignSync File-Based Object (File-based)

```
This example displays how long, in seconds, top.v has been in your
work area:

 set now [clock seconds]
 set objfetchtime [url fetchtime top.v]
 if { $objfetchtime == 0 } {
     puts "top.v is a reference."
 } else {
     set duration [expr $now - $objfetchtime]
     puts "top.v fetched to this directory $duration seconds ago."
 }
```

url filter

url filter Command

NAME

```
url filter          - Returns the persistent filter for a workspace
                      module
```

DESCRIPTION

```
This command returns the persistent filter for a workspace module.
If there is no persistent filter set (including if run on a
DesignSync object or DesignSync folder or module member or other
illegal argument type), returns a null value ("").
```

SYNOPSIS

```
url filter [-filter | -hreffilter [-all]] [--] <argument>
```

ARGUMENTS

- Workspace Module

```
Specifies one of the following arguments:
```

Workspace Module

```
<workspace module>  Specifies the workspace module for which you want
                    the persistent filter.
```

OPTIONS

-all

```
-all                The -all option is used with -hreffilter switch to
                    return the values of both simple and hierarchical
                    href filters in a comma-separated list.  If the
                    -all is not provided, only simple hierarchical
                    href values are returned.
```

-hreffilter

```
-hreffilter         The -hreffilter indicates that the hreffilter value is
                    returned.

                    This option is mutually exclusive with -filter.
```

-filter

```
-filter             The -filter option indicates that the filter value is
                    returned. (Default)

                    This option is mutually exclusive with -hreffilter.
```

--

```
   [--]             Indicates that the command should stop looking for
                    command options. Use this option when arguments
                    to the command begin with a hyphen (-).
```

RETURN VALUE

```
For workspace modules (/home/tachatterjee/MyMod/Module1),
returns the persistent filter.
```

```
If the command fails, it returns an appropriate error.
```

EXAMPLES

```
This example shows a module where a filter has been applied to the
workspace so only HTML files are populated.  Note the construction of
the filter ensures that all HTML are populated regardless of which
subdirectory or submodule they are present in.

dss> url filter Chip%0
+.../*.html
```

url getprop

url getprop Command

NAME

```
url getprop          - Retrieves a property of an object
```

DESCRIPTION

- Notes for Modules (Module-based)

```
This command retrieves properties that were previously set with 'url
setprop'. You can use 'url getprop' to access the "type" and "locked"
properties of revision control objects; however, you cannot use
'url getprop' to access all of the special, built-in properties as
returned by the 'url properties' command for objects other than
notes, notetypes, users, and project configurations. For example,
you cannot determine when an object was locked by using 'url getprop'
of the property "locktime".

Both the object and property must exist. For a note system URL, this
command always throws NO_SUCH_PROP. For a note type URL, this
command returns the default value for that property on the note type.

You can use 'url getprop' with any object type. However,
depending on the type, the command may only work in server-side
scripts, such as when accessing notes.

DesignSync automatically determines the data type of an object.
You can get the datatype assigned by DesignSync using the
'url getprop' command. You can also use the 'url setprop' command to
change the datatype of an existing object. See 'url setprop' and 'ci'
commands for more information.
```

```
Note: If the URL provided for the argument has a non-numeric
      extension, the url getprop command identifies the object as
      a branch and not a version.
```

Notes for Modules (Module-based)

```
You can use 'url getprop' to access the "basedir" to determine the
path of a workspace module.
```

SYNOPSIS

```
url getprop [--] <argument> <propertyName>
```

ARGUMENTS

- Module (Module-based)
- Module Member (Module-based)
- DesignSync Object (File-based)

```
Specifies one of the following arguments:
```

Module (Module-based)

```
<module>            Specifies the module for which you want the
                    properties previously set by the 'url
                    setprop' command.
```

Module Member (Module-based)

```
<module member>     Specifies the module member for which you
                    want the properties previously set by the 'url
                    setprop' command.
```

DesignSync Object (File-based)

```
<DesignSync object> Specifies the DesignSync object for which you
                    want the properties previously set by the 'url
                    setprop' command.
```

OPTIONS

- --

--

```
--                  Indicates that the command should stop looking for
                    command options. Use this option when arguments
                    to the command begin with a hyphen (-).
```

OPERANDS

- Object
- Property Name

Object

```
<object>            A valid object URL.
```

Property Name

```
<propertyName>      The name of a property to retrieve from the
                    object.
```

RETURN VALUE

```
For all valid arguments, returns the value set for the specified
user-defined property as a string.  also returns the values for the
built-in 'type' and 'locked properties.

For other objects: Raises error.
```

SEE ALSO

```
note getprop, url setprop, url properties, note setprops
```

EXAMPLES

- Example of Getting the DataType Property of a Module Member (Module-based)
- Example of Getting the Various Propreties of a Module (Module-based)
- Example of Getting a User Defined Property for Use in a Script (File-based)

Example of Getting the DataType Property of a Module Member (Module-based)

```
This example uses 'url getprop' command to get the DataType
property of module member File2.txt:
```

Example of Getting the Various Propreties of a Module (Module-based)

```
This example uses 'url getprop' command to get the various properties
set on module Module1:

  stcl> url getprop Module1 version
  1.9
  stcl> url getprop Module1 branch
  Property not found: branch
  stcl> url getprop Module1 hrefmode
  normal
  stcl> url getprop Module1 selector
  Trunk:
  stcl> url getprop [url vault File2.txt] DataType
  ascii
  stcl> url getprop Module1%2 basedir
  /home/tachatterjee/MyMod/Module1%2
```

Example of Getting a User Defined Property for Use in a Script (File-based)

```
This example server-side script sets and displays a CcList
user-defined property on a vault.

 url setprop "sync:///Projects/myproj/foo.v;" CcList {sal mark}
 puts [url getprop "sync:///Projects/myproj/foo.v;" CcList]
```

url inconflict

url inconflict Command

NAME

```
url inconflict      - Checks if a file merge had conflicts
```

DESCRIPTION

```
This command checks whether a merge (see the -merge option for the
populate and co commands) resulted in conflicts (returns 1) or not
(returns 0). You must resolve merge conflicts before you can check
```

```
in the file. The conflicts are considered resolved when the
file no longer contains any of the conflict delimiters (exactly 7
less-than, greater-than, or equal signs starting in column 1).
```

SYNOPSIS

```
url inconflict [--] <argument>
```

ARGUMENTS

- Workspace Module (Module-based)
- Module Member (Module-based)
- DesignSync Object (File-based)

```
Specifies one of the following arguments:
```

Workspace Module (Module-based)

```
<workspace module>  Specifies the workspace module for which you want
                    to know the conflict status.
                    Returns 0 as modules cannot be inconflict.
```

Module Member (Module-based)

```
<module member>     Specifies the module member for which you want to
                    know the conflict status.
```

DesignSync Object (File-based)

```
<DesignSync object> Specifies the DesignSync object for which you want
                    to know the conflict status.
```

OPTIONS

- --

--

```
--              Indicates that the command should stop looking for
                command options. Use this option when the argument
                to the command begins with a hyphen (-).
```

RETURN VALUE

```
For a client-side versionable object (Asic/x.v): Returns 1 (Tcl TRUE)
if there is a conflict after a merge; returns 0 (Tcl FALSE)
otherwise.

For any invalid arguments or objects that do not exist, the command
returns 0.
```

SEE ALSO

```
url modified
```

EXAMPLES

Example Showing The Status of File Merges for a Module (Module-based)

```
This example uses 'url inconflict' to get the status of file
merges in module Module1. In this example, there is no conflict.

  stcl> url inconflict sync://srv2.ABCo.com:2647/Modules/Mod/Module1
  0
```

Example Showing the Merge Status of a Module Member (Module-based)

```
This example uses 'url inconflict' to get the status of module
member File1.txt.  In this example, there is a conflict.

  stcl> url inconflict sync://srv2.ABCo.com:2647/Modules/JitaMod1/
       File1.txt
  1
```

Example Showing the Merge Status of a File-Based Object (File-based)

```
This example demonstrates how 'url inconflict' identifies a file
with conflicts.

dss> url inconflict top.v
```

```
0
dss> ci -nocomment -keep top.v

Beginning Check in operation...

Checking out: top.v     : Failed:som: Error 105: Newer version
exists in the vault.  Use '-skip' to skip the newer version, or
'co -merge' to merge with it.

dss> co -merge -nocomment top.v

Beginning Check out operation...

Success - Checked Out with conflicts version: 1.2
dss> url inconflict top.v
1
```

url leaf

url leaf Command

NAME

```
url leaf            - Returns the leaf of the URL
```

DESCRIPTION

```
Returns the leaf of the URL. The leaf is the text that
follows the last separator.

Note: In stcl/stclc mode, when specifying a URL that contains a
semicolon (;), surround the URL with double quotes.
```

SYNOPSIS

```
url leaf [--] <argument>
```

ARGUMENTS

```
Specifies one of the following arguments:
```

OPTIONS

- --

--

```
--              Indicates that the command should stop looking for
                command options. Use this option when the argument
                to the command begins with a hyphen (-).
```

RETURN VALUE

```
For a client-side versionable object (Asic/x.v): Returns a string
containing the leaf of its path (x.v).

For a client-side folder (Asic): Returns the leaf of its path
(Asic).

For a vault ("sync://holzt:2647/Projects/Asic/x.v;"): Returns
the vault name (x.v;).

For a version ("sync://holzt:2647/Projects/Asic/x.v;1.1",
"sync://holzt:2647/Projects/Asic/x.v;yellow"): Returns
object and version (x.v;1.1).

For a branch ("sync://holzt:2647/Projects/Asic/x.v;1.5.1",
"sync://holzt:2647/Projects/Asic/x.v;Golden:Latest"):
Returns object and branch name (x.v;1).

For a project (sync://holzt:2647/Projects/Asic): Returns the
project name: (Asic).

For a configuration (sync://holzt:2647/Projects/Asic/Sub@Rel1):
Returns the configuration name: (sub@Rel1).

For a server-side note type URL (sync:///Note/SyncNotes/HW-Defect-1):
Returns note type (HW-Defect-1).

For a server-side note URL (sync:///Note/SyncNotes/HW-Defect-1/1):
Returns note ID (1).

For other objects: Returns argument provided.

Note: 'url leaf' does not verify that the object exists.
```

SEE ALSO

```
url path
```

EXAMPLES

```
This example extracts the leaf "ASIC" from a URL.
  dss> url leaf sync://dvorak:2647/Projects/ASIC
  ASIC
```

url locktime

url locktime Command

NAME

```
url locktime        - Returns when a branch was locked
```

DESCRIPTION

```
This commands returns when the branch associated with the specified
object was locked. Specify a local object or a branch as the
argument.  If you specify a local object, 'url locktime' determines
the current branch for the object.

One application for this command is to determine if any
branches have been locked too long based on a project team's
design management policies. You might trigger email reminders
or perform unlock operations on objects that have been locked
too long.
```

SYNOPSIS

```
url locktime [--] <argument>
```

ARGUMENTS

- Module Object (Module-based)
- Module Member (Module-based)
- DesignSync Object (File-based)

```
Specifies one of the following arguments:
```

Module Object (Module-based)

```
<module>            Specifies the workspace module branch for which you
                    the lock time, if it is locked.
```

For server modules, you can get the lock time for a particular module branch or version. For a server module as a whole, the command returns 0, as a server module cannot be locked.

Module Member (Module-based)

<module member>	Specifies the module member for which you want the lock time, if it is locked. Always return 0 as module members are not locked in their own right. They are always locked in the context of a particular module branch. Use the showlocks command to obtain information on about member locks.

DesignSync Object (File-based)

<DesignSync object>	Specifies the DesignSync object for which you want the associated branch or version locktime.

OPTIONS

- --

--

--	Indicates that the command should stop looking for command options. Use this option when the object you specify begins with a hyphen (-).

RETURN VALUE

For all valid objects, returns the lock time, in time_t format; or, if the object is not locked, returns 0. For any invalid arguments, returns 0.

For any non-existant objects, returns error.

Note: The time_t format is the number of seconds since the birth of UNIX -- January 1, 00:00:00, 1970 (GMT). You can use the Tcl 'clock format' command to convert the time_t format to a date string.

SEE ALSO

```
url fetchtime, url properties
```

EXAMPLES

Example of Viewing the Locktime of Server Module Version (Module-based)

```
This example uses 'url locktime' to see the locktime of a module
branch of Module1 on the server:
  stcl> url locktime \
     "sync://srv2.ABCo.com:2647/Modules/Mod/Module1;1.6.1"

  1165595391
```

Example of Viewing the Locktime of a Workspace Module (Module-based)

```
This example uses a workspace module to see the locktime of the
module branch in the example above:
  stcl> url locktime Module1%1
  1165595391

Note: Workspace module Module1%1 is populated with module branch
      sync://srv2.ABCo.com:2647/Modules/Mod/Module1;1.6.1"
```

Example of Viewing the Locktime of a File (File-based)

```
This example notifies the locker of top.v if top.v has been locked
for more than a week:

  # Compute the number of seconds in a week
  set week [expr 60 * 60 * 24 * 7]
  # Find out who the locker is
  url properties top.v props
  set locker $props(locked)
  # Is the file locked?
  # Could also check whether the file is locked by using
  #   'url locktime' itself -- it returns '0' if the file is not
  #   locked.
  if { $locker != 0 } {
      # Figure out how long the file has been locked
      set now [clock seconds]
```

```
      set objlocktime [url locktime top.v]
      # If more than a week, send email
      if { $now - $objlocktime > $week } {
          #
          # Provide command to send mail here
          #
          puts "Sent mail to $locker."
      } else {puts "top.v hasn't been locked too long."}
  } else {puts "top.v is not locked."}
```

```
Note that if the branch is not locked, then objlocktime is 0, so
'now - objlocktime' is very large and does not convey the right
information. This example therefore uses 'url properties' to first
determine if the branch is locked before calling 'url locktime'.
```

url members

url members Command

NAME

```
url members          - Returns the members of the specified collection
```

DESCRIPTION

- Notes for Modules (Module-based)

```
This command returns the list of members for the specified
collection object. Specify the collection as a URL or path.
DesignSync currently supports the following collection object type:
  - Cadence cell views
    The members of a Cadence cell view collection object are
    determined by the Cadence software.
  - Synopsys cell view collections.
  - Custom generic collections (CTP collections).

This command can return the list of members with full (absolute)
paths or paths relative to the collection

When in stcl/stclc mode only, you can optionally specify Tcl variable and
code arguments. When specified, the command iterates through the
returned list of member objects (see Examples).

This command supports the command defaults system.
```

Notes for Modules (Module-based)

```
This command is for members of collections, not to get the list of
members of a module
```

SYNOPSIS

```
url members -[no]relative [--] <collection> [<varname> <code>]
```

OPTIONS

- -[no]relative
- --

-[no]relative

```
-[no]relative   Indicates whether members are displayed using a
                relative or absolute path.

               -norelative displays the members using an absolute
                path (Default).

                -relative displays the output of the command as the
                 relative path.  This output is useful for
                 identifying the collection cell view version of a
                 member for comparing against a different member
                 version.
```

--

```
--              Indicates that the command should stop looking for
                command options. Use this option when an argument
                to the command begins with a hyphen (-).
```

RETURN VALUE

```
For a collection specified as a URL or path
(file:///home/projLeader/ttlLib/and2/symbol.sync.cds,
/home/projLeader/ttlLib/and2/symbol.sync.cds):
Returns a list of URLs of view members
(file:///home/projleader/ttlLib/and2/symbol/symbol.cdb
file:///home/projleader/ttlLib/and2/symbol/pc.db
file:///home/projleader/ttlLib/and2/symbol/master.tag)

For other objects: Returns an empty list.
```

EXAMPLES

```
This example returns the members of the symbol.sync.cds Cadence
cell view:
  stcl> url members symbol.sync.cds
  file:///home/goss/Projects/Cadence/smallLib/and2/symbol/symbol.cdb
  file:///home/goss/Projects/Cadence/smallLib/and2/symbol/master.tag
  file:///home/goss/projects/Cadence/smallLib/and2/symbol/pc.db
  ...
```

url mirror

url mirror Command

NAME

```
url mirror          - Returns the URL of a local directory's mirror
```

DESCRIPTION

```
This command returns the URL of the mirror directory that was
associated with a local directory with the setmirror command. If no
mirror is set, an empty string is returned.

Note: Because of the way mirrors is configured for modules, this
command does not return any useful information in a modules
environment. For information on using mirrors in a module
environment, see the DesignSync Data Manager Administrator's Guide.
```

SYNOPSIS

```
url mirror [--] <directory>
```

OPTIONS

- --

--

```
--              Indicates that the command should stop looking for
                command options. Use this option when the argument
                to the command begins with a hyphen (-).
```

RETURN VALUE

```
For a folder with a mirror association: Returns URL of the mirror
folder.

For other objects: Returns an empty list.
```

SEE ALSO

```
setmirror
```

EXAMPLES

```
This example sets the mirror for a directory, then uses
'url mirror' to display the directory's mirror.
   dss> setmirror /home/goss/mirror /home/goss/Projects/Sportster
   Sportster   Success Set Mirror
   dss> url mirror /home/goss/Projects/Sportster
   file:///home/goss/mirror
```

url modified

url modified Command

NAME

```
url modified        - Checks if an object has been modified
```

DESCRIPTION

- Notes for Modules (Module-based)

```
This command determines whether an object has been modified since
it was fetched (returns 1) or not (returns 0).

Objects not under revision control are always flagged as
modified. Because of this behavior, 'url modified' provides a way
to determine what objects need to be checked in to preserve the
folder's current contents. Only by checking in both
revision-controlled objects that are modified and objects that
are not revision controlled will the vault contain all of the
folder's current contents. Use 'url registered' to determine whether
objects flagged as modified are under revision control.
```

Notes for Modules (Module-based)

```
Module members that have been renamed or removed are flagged as
modified, even if the contents of the object have not changed.
```

SYNOPSIS

```
url modified [--] <argument>
```

ARGUMENTS

- Workspace Module (Module-based)
- Module Member (Module-based)
- DesignSync Object (File-based)

```
Specifies one or more of the following arguments:
```

Workspace Module (Module-based)

```
<workspace module>  Specifies the workspace module for which you want
                    to find modified status.
                    Note: A workspace module is considered modified
                          if any module member has been edited or
                          a new member has been added but not yet
                          checked into the vault.
```

Module Member (Module-based)

```
<module member>     Specifies the module member for which you want
                    to find modified status.
                    Note: A module member is considered modified if
                          it has been touched. A module member is
                          also considered modified if it is added,
                          removed or renamed (moved) in the workspace
                          but not checked into the vault.
```

DesignSync Object (File-based)

```
<DesignSync object> Specifies the DesignSync object for which you want
                    to find modified status. Returns 1 if the object has
```

been modified since it was fetched from the vault. Else returns 0.

OPTIONS

- --

--

--	Indicates that the command should stop looking for command options. Use this option when the argument to the command begins with a hyphen (-).

RETURN VALUE

For any valid object, returns 1 (Tcl TRUE)) if the object has been modified or is not under revision control. If the object has not been modified, it returns 0 (tcl FALSE).

For any non-applicable or non-existant object, returns 0.

SEE ALSO

url exists, url registered, url inconflict

EXAMPLES

- Example Showing If the Module in the Workspace is Modified (Module-based)
- Example Showing If the Module Member in the Workspace is Modified (Module-based)
- Example Showing If the Files in the Workspace are Modified (File-based)

Example Showing If the Module in the Workspace is Modified (Module-based)

This example uses 'url modified' to see if a workspace module has been modified:

```
stcl> url modified Module1%0
1
```

Example Showing If the Module Member in the Workspace is Modified (Module-based)

This example uses 'url modified' to see if a module member has been modified:

```
stcl> url modified File1.txt
0
```

Example Showing If the Files in the Workspace are Modified (File-based)

```
This example shows various results using 'url modified' and 'url
registered':

  dss> ls -report ORS top*
  Object Type      Version                    Status  Name
  -----------      -------                    ------  ----
  File             1.2              Locally Modified  top.gv
  File             1.2 -> 1.3             Up-to-date  top.v
  File             Unmanaged                       -  top.log
  Referenced File  Refers to: 1.1         Up-to-date  top.f
  dss> url modified top.gv
  1
  dss> url modified top.v
  0
  dss> url modified top.log
  1
  dss> url registered top.v
  1
  dss> url registered top.log
  0
```

url naturalpath

url naturalpath Command

NAME

```
url naturalpath     - Returns the natural path for a module member
```

DESCRIPTION

```
This command is run against module member objects in a workspace to
provide the natural path for the specified object.  The natural path
is the location of the object under the module base directory.

Note: This command also provides the natural path for objects that
have been added to a module, but have not yet been checked in.
```

SYNOPSIS

```
url naturalpath [--] <argument>
```

ARGUMENTS

- Workspace Module Member

Workspace Module Member

<Workspace module member>	Specify a module object to determine the natural path of the module member. Note: The module member cannot be a module folder.

OPTIONS

- --

--

--	Indicates that the command should stop looking for command options. Use this option when the argument to the command begins with a hyphen (-).

RETURN VALUE

For a module member:	returns the natural path of module member. If a module member has been moved, it returns the new location of the module member. If a module member has been removed, but kept in the workspace, the original natural path is reported until the module has been checked in.
For any other values:	Not applicable; returns an empty list("").

SEE ALSO

```
add
```

EXAMPLES

- Example Showing the Natural Path of a Module Member

Example Showing the Natural Path of a Module Member

```
The following example returns the natural path of the chipdoc.txt
file, located in module instance Chip%0 with a base directory of
/home/rsmith/MyModules/chip.

stcl> url naturalpath chipdoc.txt
 /doc/chipdoc.txt
```

Example Showing Using the Natural Path to Unlock a Module Member

```
The following example shows how to use url naturalpath, along with
url vault and url branchId, to perform an unlock command.  The
code fragment unlocks a file.  This fragment assumes the module
instance name was previously passed to the mod variable and the
desired filename, the same file used in the previous example, was
passed to the modfile variable.

stcl> unlock -modulecontext [url vault $mod]\;[url branchid $mod] \
 [url naturalpath $modfile]

Beginning Unlock operation...

Unlocking:   sync://srvr2.ABCo.com:2647/Modules/Chip;1 :
    /doc/chipdoc.txt: Unlocked

Unlock operation finished.

 {Objects succeeded (1)} {}
```

url notes

url notes Command

NAME

```
url notes           - Returns the notes attached to the specified
                      object
```

DESCRIPTION

```
This command gathers a list of notes attached to a DesignSync object
or a server module (branch or version). Because notes are objects
```

addressed by URLs, this command returns a list of URLs. The list may be filtered by note type and by a specific set of query criteria on the note type. When applied to RevisionControl notes, the url notes command searches the Objects field.

This command is a wrapper for: note query [-type <notetype> -attached <ObjectUrl> -dbquery <Query>. In most cases, the note query command provides superior capabilities.

The url notes command' is server-side only. For more information, see the "server-side" and "rstcl" help topics.

SYNOPSIS

```
url notes [-type <type> [-dbquery <query>]] [--]
          <argument>
```

ARGUMENTS

- Server Module Version (Module-based)
- DesignSync File (File-based)

Specifies one of the following arguments:

Server Module Version (Module-based)

<server module>	Specifies the server module or module branch or module version to which the notes are attached. If no notes are attached, returns an empty list

DesignSync File (File-based)

<DesignSync object>	Specifies the DesignSync object to which the notes are attached. If no notes are attached, returns an empty list.

OPTIONS

- -type
- -dbquery
- --

-type

```
-type <type>          The name of a note type, which must exist, to
                      query against.
```

-dbquery

```
-dbquery <query>      A valid dBase query string, used to further
                      constrain the set of notes returned.
```

--

```
--                    Indicates that the command should stop looking
                      for command options. Use this option when the
                      argument to the command begins with a hyphen (-).
```

OPERANDS

- Object

Object

```
<object>            A valid object URL.
```

RETURN VALUE

```
For any valid object, returns a list of note URLs; or, if there are
no notes attached, an empty list.

For any invalid object, returns an error.

For any non-existent object, returns an empty list.
```

SEE ALSO

```
note links, note query, url contents
```

EXAMPLES

- Example Showing the List of Specific Note Types in a Specific State
- Example of a Script Fragment that Extracts Attached Note Information

Example Showing the List of Specific Note Types in a Specific State

```
The following example returns the list of SyncDefect notes in the
open state attached to the Munich project:

set notes [url notes
  sync:///Projects/Munich -type SyncDefect -dbquery "State='open'"]
```

Example of a Script Fragment that Extracts Attached Note Information

```
The following excerpt of a server-side script extracts all of the
notes in all of the projects on a SyncServer and prints their
titles.  The excerpt uses the 'url notes' command to extract
the notes of a project.

foreach project [url projects sync:///] {
  foreach note [url notes $project] {
     puts <pre>
     puts "Project: $project"
     puts "NoteURL: $note"
     puts "Notetype: [url leaf [url container $note]]"
     puts "Note Id: [url leaf [url path $note]]"
     puts "Note Title: [note getprop $note Title]"
     puts </pre>
  }
}
```

url owner

url owner Command

NAME

```
url owner           - Returns the owner of an object
```

DESCRIPTION

```
This command returns the owner of the specified object. The object
can be a project, project configuration, branch, or vault.

The owner of a branch is the creator of the initial version of the
branch unless a different owner has been specified with the
setowner command. For example, the default owner of the Trunk branch
(branch 1) is the creator of version 1.1  The owner of a design
object's vault is defined as the owner of the object's Trunk branch.

The 'url properties' command also returns an object's owner. Use
```

```
'url properties' when you need more property information than just
the object owner.
```

SYNOPSIS

```
url owner [--]  <argument>
```

ARGUMENTS

- Workspace Module (Module-based)
- Module Member (Module-based)
- DesignSync Object (File-based)

```
Specifies one of the following arguments:
```

Workspace Module (Module-based)

```
<workspace module>  Specifies the module or module branch or module
                    version for which you want to know the owner.

                    Note: For a module, the owner is the person who
                    creates the module. Since creating a module creates
                    version 1.1, the owner of the module, branch 1 and
                    version 1.1 is always the same person.
```

Module Member (Module-based)

```
<module member>     Specifies the module member for which you want to
                    know the owner.
                    Note: The 'url owner' command can only be run on
                          workspace module members.
```

DesignSync Object (File-based)

```
<DesignSync object> Specifies the DesignSync object for which you
                    want the owner.
```

OPTIONS

- --

--

```
--                  Indicates that the command should stop looking for
                    command options. Use this option when the argument
                    to the command begins with a hyphen (-).
```

RETURN VALUE

```
For all valid objects, returns the username of the object owner.

For all invalid or non-existent objects, returns an applicable error.
```

SEE ALSO

```
setowner, url properties
```

EXAMPLES

- Example Showing The Owner of a Module (Module-based)
- Example Changing the Owner of a File-Based Object (File-based)
- Example of Returning the Owner of the File-based Project (File-based)

Example Showing The Owner of a Module (Module-based)

```
This example uses 'url owner' to get the username of the
owner of a module:
  stcl> url owner "sync://srv2.ABCo.com:2647/Modules/Module1"
  tachatterjee
```

Example Changing the Owner of a File-Based Object (File-based)

```
This example returns the owner (barbg) of the main branch of
reg5.v. The owner is then changed to 'goss' and the change is
verified. Note that the second 'url owner' command passes the
vault as the argument, which shows that the vault owner is always
the owner of the main branch.
  stcl> url owner "sync://holzt:2647/Projects/Sportster/decoder/
        reg5.v;1"
  barbg
  stcl> setowner "sync://holzt:2647/Projects/Sportster/decoder/
        reg5.v;1"  goss
  Success:   setowner
  stcl> url owner [url vault reg5.v]
  goss
```

Example of Returning the Owner of the File-based Project (File-based)

```
This example returns the owner of a ProjectSync project called
ASIC, and the owner of the Alpha configuration.

  stcl> url owner  "sync://vonnegut:2647/Projects/ASIC"
  mmf
  stcl> url owner  "sync://vonnegut:2647/Projects/ASIC@Alpha"
  wayne
```

url path

url path Command

NAME

```
url path            - Extracts the path section of a URL
```

DESCRIPTION

- Module Notes (Module-based)

```
This command extracts the path section of a URL, stripping
off the protocol and machine name. This command also returns the
absolute path of an object when a relative path is specified.

Note: In stcl/stclc mode, when specifying a URL that contains a
semicolon (;), surround the URL with double quotes.

On Windows, this command returns a localized path using
"\" characters, instead of "/" characters. Use the following
Tcl example to reverse the "\" characters:

  stcl> url path .
  e:\build\main\src\doc\
  stcl> join [split [url path .] \\] /
  e:/build/main/src/doc
```

Module Notes (Module-based)

```
NOTE: To find the base directory of a module instance object,
use the 'url getprop' command.
```

SYNOPSIS

```
url path [--] <object>
```

OPTIONS

- --

--

```
--              Indicates that the command should stop looking for
                command options. Use this option when the argument
                to the command begins with a hyphen (-).
```

RETURN VALUE

```
For all valid objects, returns the absolute path without URL protocol
or host information, (for example, /home/karen/Asic/x.v).

For other objects, including non-existent objects: Returns absolute
path of the current directory with the object concatenated; for
example, 'url path' applied to a nonexistent file named 'junk'
returns /home/karen/junk. The 'url path' command does not verify that
the object exists.
```

SEE ALSO

```
url leaf, url getprop
```

EXAMPLES

- Example Showing How to Get Path and Reverse the Separator
- Example Showing the URL Path of the Server Module (Module-based)
- Example Showing the URL Path of a File-based Server Object (File-based)
- Example Showing the URL Path of a File-based Workspace Object (File-based)

Example Showing How to Get Path and Reverse the Separator

```
This example shows how to reverse the "\" characters on Windows:
  stcl> url path .
  e:\build\main\src\doc\
  stcl> join [split [url path .] \\] /
```

```
  e:/build/main/src/doc
```

Example Showing the URL Path of the Server Module (Module-based)

```
This example uses 'url path' to get the path of a server module:
  stcl> url path sync://srv2.ABCo.com:2647/Modules/Module1
 /Modules/Module1
```

Example Showing the URL Path of a File-based Server Object (File-based)

```
This example extracts the path "/Projects/ASIC/" from a URL.
  dss> url path sync://dvorak:2647/Projects/ASIC
  /Projects/ASIC
```

Example Showing the URL Path of a File-based Workspace Object (File-based)

```
This example returns the full path of top.v, which is in the
current working directory:
  dss> url path top.v
  /home/Projects/Sportster/synth
```

url projects

url projects Command

NAME

```
url projects        - Returns a SyncServer's ProjectSync projects
```

DESCRIPTION

```
This command returns the list of all ProjectSync projects defined
on the specified SyncServer. See ProjectSync User's Guide for details
on creating projects. Note that:
o ProjectSync projects may or may not have an associated
  DesignSync vault. You specify whether the project has an
  associated vault when you create the project.
o Creating a DesignSync project vault, even when located in a
  SyncServer's Projects directory, does not automatically create a
  ProjectSync project. You must independently create the
  ProjectSync project and associate it with the DesignSync vault.
```

The 'url projects' command accepts one argument, a sync: URL. When run from a server-side script, the projects defined on the SyncServer running the script are returned (the specified argument is always mapped to sync:///Projects). When run from a client, the projects defined on the SyncServer specified by the URL are returned (the specified argument is always mapped to sync://<host>:<port>/Projects). When run from the client or server, 'url projects' returns an empty string if you specify a file: URL.

SYNOPSIS

```
url projects <object>
```

OPTIONS

none

RETURN VALUE

For all valid objects, returns the projects defined for the specified server. In server-side scripts, 'url projects' returns the projects defined on the server on which the script is running.

For non-applicable or non-existent objects, returns empty list.

SEE ALSO

url configs, url users

EXAMPLES

This example shows two methods for returning the ProjectSync projects on the holzt:2647 SyncServer.

```
stcl> url projects sync://holzt:2647
sync://holzt:2647/Projects/Sportster sync://holzt:2647/Projects/Test
stcl> url projects [url vault top.v]
sync://holzt:2647/Projects/Sportster sync://holzt:2647/Projects/Test
```

url properties

url properties Command

NAME

```
url properties      - Returns properties for the specified object
```

DESCRIPTION


```
This command retrieves all the properties of the specified object
and returns the values in a Tcl array passed by name. The Tcl array
need not exist prior to the call. If the array does exist, its
contents are first emptied and then filled in with the property
data for the object. If <varname> was previously set as a scalar
variable, it is changed to an array by this command. If the command
encounters an error, <varname> is left unset, regardless of its
prior state. The Tcl array is indexed by property name.

The properties defined for an object depend on the object's type:

note        - The current property values on the note.
note type   - The default property values of the note type.
note system - An empty set.
user        - The fixed set of properties of the user profile:
              EmailAddr, Key, Name, PageNumber, PhoneNumbr, UserList
              and Username. For backward compatibility, the shadow
              properties email, name, pager, phone, and userName
              are also returned.
```

Properties Associated with Module Objects (Module-based)

```
The properties on an object can be:

name        - The name of the specified object.
description - The generic description for the object, or an empty
              string if none exists.
type        - The type of the specified object. Examples are
              File, Folder, Vault, Version, Branch, Project,
              and Project Configuration.
              Note: There may be other types present as a result
                    of using the CustomType System, DesignSync DFII
                    or DesignSync Custom Compiler.
owner       - The owner of the object. The following object types
              have owners: modules, module folders, module members,
              module versions,and module branches. If owner is the
              only property you are interested in, use 'url owner'.
locked      - The name of the user who has the object locked, or '0'
```

```
                if it is unlocked. A non-zero value can be expected only
                for files, branches, and versions. Specifying a file
                has the same effect as specifying the file's current
                branch to the command.
locktime      - The time, in time_t format, that the object was
                locked (if the object is locked -- value of 'locked'
                property is nonzero), otherwise '0'. If locktime is the
                only property you are interested in, use 'url
                locktime'. Note that you can convert the time_t
                format to a date string using the Tcl 'clock format'
                command.
citime        - The time, in time_t format, that a version was created
                in the vault. This time is not influenced by the
                "-retain" option to ci/co/populate; citime is always
                the actual time the version was created. Note that you
                can convert the time_t format to a date string using
                the Tcl 'clock format' command.
log           - The log information for the specified object. If the
                object is a version, its checkin log is returned,
                unless it is a placeholder (upcoming) version, in
                which case its checkout log is returned. If the
                object is a file, its ongoing log is returned.
selector      - The selector list (tag) associated with a ProjectSync
                project configuration that identifies the versions of
                DesignSync data that are part of the configuration.
exposure      - The list of team members (usernames) associated with a
                project configuration. The configuration owner is
                always included in the exposure list. Note that if the
                member list is the default of all users defined
                on the SyncServer, then the exposure list is empty.
parents       - The parent workspace(s) of the object. The parent
                workspace is the base directory of other modules in the
                workspace containing an href to specified module
                argument. The value is space delimited tcl list showing
                the module instance name followed by the workspace base
                directory.
moduleviews   - The list of persistent module views set on the module
                workspace. This property only exists if a persistent
                module view has been set on the workspace. If a view
                has been set and cleared, the returned value is an
                empty string ("").

Additional properties on a module that has been moved (with the
exportmod/importmod commands are:)

SyncImportedURL      - The URL of the original module location.

SyncImportedBackRefs - The back references contained within the
                       original module.

Note that you use 'url properties' to access predefined (built-in)
properties. To access user-defined properties, as created
by 'url setprop', use 'url getprop'. You cannot use 'url setprop'
to modify these built-in properties.
```

Properties of File Objects (File-based)

```
The properties on an object can be:

name        - The name of the specified object.
description - The generic description for the object, or an empty
              string if none exists.
type        - The type of the specified object. Examples are
              File, Folder, Vault, Version, Branch, Project,
              and Project Configuration.
              Note: There may be other types present as a result
                    of using the CustomType System, DesignSync DFII
                    or DesignSync Custom Compiler.
owner       - The owner of the object. The following object types
              have owners: projects, project configurations,
              vaults, and branches. If owner is the only
              property you are interested in, use 'url owner'.
locked      - The name of the user who has the object locked, or '0'
              if it is unlocked. A non-zero value can be expected only
              for files, vaults, branches, and versions. If a vault
              is specified, the default branch is examined.
              Specifying a file has the same effect as specifying
              the file's current branch to the command.
locktime    - The time, in time_t format, that the object was
              locked (if the object is locked -- value of 'locked'
              property is nonzero), otherwise '0'. If locktime is the
              only property you are interested in, use 'url
              locktime'. Note that you can convert the time_t
              format to a date string using the Tcl 'clock format'
              command.
citime      - The time, in time_t format, that a version was created
              in the vault. This time is not influenced by the
              "-retain" option to ci/co/populate; citime is always
              the actual time the version was created. Note that you
              can convert the time_t format to a date string using
              the Tcl 'clock format' command.
log         - The log information for the specified object. If the
              object is a version, its checkin log is returned,
              unless it is a placeholder (upcoming) version, in
              which case its checkout log is returned. If the
              object is a file, its ongoing log is returned.
selector    - The selector list (tag) associated with a ProjectSync
              project configuration that identifies the versions of
              DesignSync data that are part of the configuration.
exposure    - The list of team members (usernames) associated with a
              project configuration. The configuration owner is
              always included in the exposure list. Note that if the
              member list is the default of all users defined
              on the SyncServer, then the exposure list is empty.
parents     - The parent workspace(s) of the object.

Note that you use 'url properties' to access predefined (built-in)
properties. To access user-defined properties, as created
by 'url setprop', use 'url getprop'. You cannot use 'url setprop'
to modify these built-in properties.
```

SYNOPSIS

```
url properties [--] <argument> <array_name>
```

ARGUMENTS

- Module (Module-based)
- DesignSync Object (File-based)
- Array Name

```
Specify the following arguments:
```

Module (Module-based)

```
<module>            Specifies the module for which you want the
                    predefined properties.
                    For a server module or a server module branch or a
                    server module version, this information is similar
                    to the information about a DesignSync vault.
                    For a workspace module, the information can contain
                    additional property information.
```

DesignSync Object (File-based)

```
<DesignSync object> Specifies the DesignSync object for which you want
                    the predefined properties. The  properties are
                    returned in a Tcl array passed by name.
```

Array Name

```
<array_name>        The name of a Tcl variable in which to store the
                    property values returned.
```

OPTIONS

- --

--

```
 --                  Indicates that the command should stop looking for
                     command options. Use this option when the argument
                     to the command begins with a hyphen (-)._
```

RETURN VALUE

```
 Returns the property values indicated in the supplied array variable:

 For a client-side versionable object (Asic/x.v): Returns these property
 values in the supplied array variable: name, type, locked, locktime,
 citime, and log.

 For a client-side folder (Asic/Sub): Returns these property values:
 name and type.

For a server-side note type URL (sync:///Note/SyncNotes/HW-Defect-1):
 Returns the properties of the note type.  Values are only listed for
 those properties that have default values specified in the note type
 definition.

 For a server-side note URL (sync:///Note/SyncNotes/HW-Defect-1/1):
 Returns the properties of the note type, as well as the values set
 for those properties.

 For a user URL (sync:///Users/chris): Returns the property values
 set for that user.

 For a version ("sync://holzt:2647/Projects/Asic/x.v;1.1",
 "sync://holzt:2647/Projects/Asic/x.v;yellow"): Returns
 these property values: name, type, locked, locktime, citime,
 and log.

 For a branch ("sync://holzt:2647/Projects/Asic/x.v;1.5.1",
 "sync://holzt:2647/Projects/Asic/x.v;Golden:Latest"):
 Returns these property values: name, type, owner, locked, locktime,
 citime, and log.

 For a vault ("sync://holzt:2647/Projects/Asic/x.v;"): Returns these
 property values: name, type, owner, locked, locktime, citime, and
 log.

 For a project (sync://holzt:2647/Projects/Asic): Returns these
 property values: name, description, type, and owner.

 For a configuration ("sync://holzt:2647/Projects/Asic/Sub@Rel1"):
 Returns these property values: name, description, type, owner,
 selector, and exposure.

 For any invalid object, returns an appropriate error.
```

SEE ALSO

```
 note getprop, note setprops, url getprop, url setprop, url locktime,
 url owner, server-side
```

EXAMPLES

- Example Showing the Properties of a Module (Module-based)
- Example Scripts Showing a Specific Property (File-based)
- Example Script Showing All Properties of a Project (File-based)

Example Showing the Properties of a Module (Module-based)

```
 This example uses 'url properties' to get the properties of a
 module:
   url properties Indian x
foreach prop [array names x] {
    puts "prop $prop=$x($prop)<BR>"
}

  prop recursive=1<BR>
  prop type=Module<BR>
  prop basedir=/home/tachatterjee/Example<BR>
  prop description=<BR>
  prop txnuid=00000000000000000000000000000000<BR>
  prop mappedpath=<BR>
  prop hrefs=NorthIndian {name {} naturalpath {} mappedpath {}
   uid 00000000000000000000000000000000
   target sync://srv2.ABCo.com:2647/Modules/Cuisines/NorthIndian
   dtarget {sync://srv2.ABCo.com:2647/Modules/Cuisines/NorthIndian;Trunk:}
   starget {sync://srv2.ABCo.com:2647/Modules/Cuisines/NorthIndian;1.1}
   hrefinstname NorthIndian modinstname NorthIndian%0
   basedir /home/tachatterjee/Example/NorthIndian
   relpath NorthIndian version {} targetsel Trunk:
   targetver 1.1 hreftype
   Module state added
   servertarget sync:///Modules/Cuisines/NorthIndian}<BR>
  prop name=Indian<BR>
  prop selector=Trunk:<BR>
  prop hrefmode=normal<BR>
  prop civ=<BR>
  prop uid=9ce32a1a95f4547039a55e7c3bd34906<BR>
  prop owner=<BR>
  prop exposure=<BR>
  prop toplevel=0<BR>
  prop hreffilter=<BR>
  prop naturalpath=<BR>
  prop mergefrom=<BR>
  prop keys=kkv<BR>
  prop parents=WorldCuisine%0{/home/tachatterjee/Example/worldcusine}
   AsianCusine%1 (/home/tachatterjee/Example/asiancusine)<BR>
  prop version=1.4<BR>
  prop filter=<BR>
  prop target=sync://srv2.ABCo.com:2647/Modules/Cuisine/Indian<BR>
```

```
 prop modinstname=Indian%0<BR>
```

Example Scripts Showing a Specific Property (File-based)

```
In this example (server-side script), each property for user
Joe Brown is displayed:

  url properties "sync:///Users/jbrown" userProps
  foreach prop [array names userProps] {
    puts "Prop $prop=$userProps($prop)<BR>"
  }

The properties that are displayed by this script are for User web
object type.

In this example (server-side script), the field and field values of
note number 3 of the BugReport notetype are returned:

  url properties "sync:///Note/SyncNotes/BugReport/3" noteProps
  foreach prop [array names noteProps] {
    puts "Prop $prop=$noteProps($prop)<BR>"
 }

The HTML page resulting from this script is:

  Prop KeyWords=Release Notes
  Prop State=new
  Prop Resp=goss
  Prop CCList=
  Prop Customer=Other
  Prop DateCreate=2003-05-29 12:34:56
  Prop Author=goss
  Prop Severity=STOPPER
  <and so on>

In this example, the properties for the user 'jbrown' are returned:

   url properties "sync:///Users/jbrown" userProps
   foreach prop [array names userProps] {
     puts "Prop $prop=$userProps($prop)<BR>"
   }

The HTML page resulting from this script is:

  Prop Key=gdyb21LQTcIANtvYMT7QVQ==
  Prop UserList=
  Prop PhoneNumbr=555-5555
  Prop InitVector=
  Prop Username=jbrown
  Prop userName=jbrown
  Prop email=jbrown@synchronicity.com
  Prop Name=Joe Brown
  Prop name=Joe Brown
  Prop phone=555-5555
```

```
  Prop EmailAddr=jbrown@synchronicity.com
  Prop PageNumber=555-6666
  Prop pager=555-6666

Notes:
  - The Key property is the user's encrypted password.
  - The InitVector property records the expiration time of a user
    profile that was created from an LDAP database.
```

Example Script Showing All Properties of a Project (File-based)

```
This server-side stcl script outputs all the properties of the Asic
project, each on its own line:

  url properties "sync:///Projects/Asic" Props
  foreach prop [array names Props] {
    puts "Prop $prop=$Props($prop)<BR>"
  }

For use on the client side, you must specify the <host>:<port>
in the URL of the project. Also, the HTML <BR> tag to control the
output formatting does not apply:

  url properties "sync://holzt:2647/Projects/Asic" Props
  foreach prop [array names Props] {
    puts "Prop $prop=$Props($prop)"
  }

For the server-side script, the results are displayed as an HTML
page in your browser. For the client-side script, the results are
output to stdout.
```

url registered

url registered Command

NAME

```
url registered      - Checks whether an object is under revision
                      control
```

DESCRIPTION

- Notes for modules (Module-based)

```
This command checks whether an object has been put under revision
control (returns 1) or not (returns 0). Objects that cannot be put
under revision control always return 0.
```

```
The url registered command looks up an object on the server to verify
that the object is under revision control.  By contrast, the -managed
option to ls command checks the workspace metadata to see if the
object is managed.
```

Notes for modules (Module-based)

```
If a module member has been added, but not checked in, the return
value for the member is 0.
```

SYNOPSIS

```
url registered [--] <argument>
```

ARGUMENTS

- Workspace Module (Module-based)
- Module Member (Module-based)
- DesignSync Object (File-based)

```
Specifies one of the following arguments:
```

Workspace Module (Module-based)

```
<workspace module>  Specifies the module for which you want to know
                    the revision control status.

                    Note: For a workspace module, it should always
                    return 1. If a workspace module is somehow removed
                    from the server, returns 0. Specifying a server URL
                    returns 0.
```

Module Member (Module-based)

```
<module member>     Specifies the module member for which you want to
                    know the revision control status.

                    Note: An object that is added to the module but not
                    yet checked into the vault returns 0.
```

DesignSync Object (File-based)

```
<DesignSync object> Specifies the DesignSync object for which you want
                    to know the revision control status.
```

OPTIONS

- --

--

```
--                  Indicates that the command should stop looking for
                    command options. Use this option when the argument
                    to the command begins with a hyphen (-).
```

RETURN VALUE

```
If the object is under revision control, returns 1 (Tcl TRUE).  If the
object is not under revision control, returns 0 (Tcl FALSE).

If the object does not exist or cannot be versioned, returns 0.
```

SEE ALSO

```
url exists, url fetchedstate
```

EXAMPLES

- Example Showing Whether a Module is Under Revision Control (Module-based)
- Example Showing Whether a Module Member is Under Revision Control (Module-based)
- Example Showing Whether a File Under Revision Control (File-based)
- Example Showing a File Deleted From the Server (File-based)

Example Showing Whether a Module is Under Revision Control (Module-based)

```
This example uses 'url registered' command to checked whether a
workspace  module is under revision control:
 stcl> url registered Module1%0
 1
```

Example Showing Whether a Module Member is Under Revision Control (Module-based)

```
This example uses 'url registered' command to checked whether
module members are under revision control.
 stcl> url registered File1.txt
 1

 stcl> url registered file1.txt
 0
```

Example Showing Whether a File Under Revision Control (File-based)

```
This example checks whether files are under revision control:

  dss> ls -report ORS top*
  Object Type      Version                    Status  Name
  -----------      -------                    ------  ----
  File             1.2              Locally Modified  top.gv
  File             1.2 -> 1.3             Up-to-date  top.v
  File             Unmanaged                       -  top.log
  Referenced File  Refers to: 1.1         Up-to-date  top.f

  dss> url registered top.gv
  1
  dss> url registered top.v
  1
  dss> url registered top.f
  1
  dss> url registered top.log
  0
```

Example Showing a File Deleted From the Server (File-based)

```
This example shows a file that has been removed on the server.
Note that ls -managed shows the file as managed, while url registered
does not.

Note:  This example shows a file removed with rmvault.  Files that
are removed from a module or retired from DesignSync are still
DesignSync objects and show as registered.

stcl> rmvault sync://tallis:30132/Projects/example/file1.txt;
 file1.txt;: Success deleted

stcl> ls -managed ./file1.txt
 Time Stamp             WS Status  Version   Type        Name
 ----------             ---------  -------   ----        ----
 09/19/2007 10:14                  1.1       Copy        file1.txt

stcl> url registered file1.txt
 0
```

url relations

url relations Command

NAME

```
url relations       - Determine collection object dependencies
```

DESCRIPTION

```
This command returns the objects on which a given collection object
depends. Specify the collection as a URL or path. The relationship
specification (relation_name argument) supported by DesignSync for
Cadence cell view collections is "dependencies". For custom generic
(CTP) collections, DesignSync supports any relation name set up for
the collection.

This command is useful for determining the entire set of files
associated with a design object. For example, you might create
an stcl script that returns all of the dependencies of a collection
object, then checks out the collection object and its dependencies.

When in stcl/stclc mode only, you can optionally specify Tcl
variable and code arguments. When specified, the command iterates
through the returned list of member objects (see Examples).

The url relations command supports the following collection objects:
 - Cadence cell views
     The 'url relations' command determines dependencies
     from the pc.db file, which is located in the cell view folder.
     Use the 'addcdslib' command to resolve dependency paths.
 - Custom generic (CTP) collections
   For these collections, DesignSync supports any relation name set
   up for the collection.
```

SYNOPSIS

```
url relations [--] <collection> <relation_name> [<varname> <code>]
```

OPTIONS

- --

--

```
--              Indicates that the command should stop looking for
                command options. Use this option when an argument
                to the command begins with a hyphen (-).
```

RETURN VALUE

```
For a collection specified as a URL or path
(file:///home/projLeader/ttlLib/and2/symbol.sync.cds,
/home/projLeader/ttlLib/and2/symbol.sync.cds):
Returns a list of URLs of objects on which this collection is
dependent as a list of lists, each sublist containing two
values: a collection URL and a string of the format "alias:name"
(file:///home/projlead/Projects/ttlLib/and2/symbol
 ttlLib:and2/symbol.sync.cds
 file:///home/projlead/Projects/ttlLib/nor2/symbol
 ttlLib:nor2/symbol.sync.cds)

If the alias is unknown, the URL is replaced by the string
"<unrecognized alias>". For Cadence cell views, use the
'addcdslib' command to resolve library paths.

If there are internal dependencies -- dependencies that point to
components within the object itself, the URL is the collection
object itself.

For an object that is not a collection: Returns an empty list.
```

SEE ALSO

```
addcdslib, url members
```

EXAMPLES

```
This command shows the dependencies of a collection object on one
or more other collection objects. Note: For Cadence cell view
collections, use the 'addcdslib' command to resolve dependency
paths. In this example, the cds.lib file in /home/Libraries
contains the library definition for "basic", but not for "sample".

 stcl> url relations cmos_sch.sync.cds dependencies
   {<unrecognized alias>} basic:vdd/symbol.sync.cds
   {<unrecognized alias>} basic:gnd/symbol.sync.cds
   {<unrecognized alias>} sample:nmos/symbol.sync.cds
 stcl> addcdslib /home/Libraries
 stcl> url relations cmos_sch.sync.cds dependencies
   file:///home/tgoss/Projects/Cadence/basic/opin/symbol.sync.cds
   basic:opin/symbol.sync.cds
   file:///home/tgoss/Projects/Cadence/basic/gnd/symbol.sync.cds
   basic:gnd/symbol.sync.cds
```

```
{<unrecognized alias>} sample:nmos/symbol.sync.cds
```

url resolveancestor

url resolveancestor Command

NAME

```
url resolveancestor - Returns the closest common ancestor of two
                      versions
```

DESCRIPTION

```
This command returns the closest common ancestor of two versions of
the same object (file, module or collection object). DesignSync uses
the closest common ancestor when merging two versions. DesignSync
compares the versions to the ancestor to determine how each version
has changed, then performs the merge. The two versions being merged
are called the "merge sides".

For the "url resolveancestor" command, one of the merge sides is
specified by the "-version <selectorList>" option. DesignSync
determines the other merge side from the object argument:
  o If the object is a local object, then DesignSync uses the
    current (last-retrieved) version, as stored in the object's
    local metadata.
  o If the object is a version, then DesignSync uses that version.
  o If the object is a branch, then DesignSync uses the Latest
    version of the object on that branch.
  o Any other object type causes DesignSync to throw an exception.

DesignSync records "merge edges" -- information about what versions
participated in the merge -- with the new version resulting from a
merge. DesignSync uses merge edges in future calculations of closest
common ancestors instead of always going back to the original
ancestor (by considering only branch points and not merge
edges). This capability relieves you from having to resolve the same
merge conflicts during future merges. Specify the -noedges option if
you want "url resolveancestor" to return the common ancestor without
considering merge edges.

Note: DesignSync does not currently record merge edges from
-overlay (without -merge) and -skip operations.
```

SYNOPSIS

```
url resolveancestor [-noedges] -version <selector>[,<selector>...]
                    [--] <argument>
```

ARGUMENTS

Specifies one of the following arguments:

Module (Module-based)

<module>	Specifies the module for which you want the closest common ancestor of two different versions. The specified module can be a workspace module or a server module. If a whole module is given, it is taken as version 1:Latest, if a module branch is given then it is taken as the Latest on that branch.

Module Member (Module-based)

<module member>	Specifies the workspace module member for which you want the closest common ancestor of two different versions. Note: The versions used are for the member vault and not the module.

DesignSync Object (File-based)

<DesignSync object>	Specifies the DesignSync object for which you want the closest common ancestor of two different versions.

OPTIONS

-noedges

-noedges	Specifies not to consider merge edges

when determining the closest common ancestor.

Note: The -noedges option applies only to merge edges created across-branches. Merge edges within a branch ("skip" edges) are still considered when computing the closest common ancestor.

-version

-version <selector> Specifies one of the versions (merge sides) to compare. See the "selectors" help topic for more information on selectors. DesignSync determines the other merge side from the command argument.

Notes:
- If you specify Latest or Date(<date>), DesignSync uses branch 1 (1:Latest,1:Date(<date>).
- To use -version to specify a branch, specify both the branch and version as follows: '<branchtag>:<versiontag>', for example, 'Rel2:Latest'. You can also use the shortcut, '<branchtag>:', for example "Rel2:". If you don't explicitly specify the branch selector in this way, DesignSync does not resolve the selector as a branch selector.

--

-- Indicates that the command should stop looking for command options. Use this option when the argument to the command begins with a hyphen (-).

RETURN VALUE

For all valid objects, returns the version number of the closest common ancestor of the current version of this object and the version specified with the -version option.

For all non-valid objects, or non-existent objects, returns an error.

Note: If two valid versions are specified, there is always a return value, because all versions have a common ancestor of version 1.1. If two valid versions are not specified, an error is raised.

SEE ALSO

```
populate, co, selectors
```

EXAMPLES


```
Assume the following version history for a file called "top.v":

           Trunk (branch 1)        Dev (branch 1.2.1)

           top.v;1.1
               |     Branch "Dev"
           top.v;1.2  ---------->  top.v;1.2.1.1
               |       Created         |
  (Bronze) top.v;1.3                 top.v;1.2.1.2
               |                       |
           top.v;1.4                 top.v;1.2.1.3
               |        Merge          |
           top.v;1.5  ---------->  top.v;1.2.1.4
               |                       |
           top.v;1.6                 top.v;1.2.1.5
                                       |
                                     top.v;1.2.1.6
                                       |
                                     top.v;1.2.1.7
```

Example Showing Common Ancestor from Versions on the Same Branch

```
Assume that version 1.3 of "top.v" is tagged "Bronze", and version
1.6, which is the Latest version on the same branch (Trunk), is the
current version:

  stcl> url resolveancestor -version Bronze top.v
  1.3

DesignSync compares the version tagged "Bronze" (1.3) and the
current version (1.6) and returns 1.3 as the closest common
ancestor. Whenever both versions are on the same branch, the lower
version number is, by definition, the closest common ancestor.
```

Example Showing Common Ancestor from Versions on Different Branches

```
Assume that your current version is 1.2.1.7. You want to know the
common ancestor of your current version and the Latest version on
```

```
the Trunk branch.

  stcl> url resolveancestor -version Trunk top.v
  1.5

DesignSync compares the Latest version on Trunk (1.6) and the
current version (1.2.1.7) and returns 1.5 as the closest common
ancestor. When versions 1.5 and 1.2.1.3 were merged to create
version 1.2.1.4, DesignSync recorded the merge edge. Version
1.2.1.4 includes the resolution of any conflicts between 1.2.1.3
and 1.5. Subsequent merges between Trunk and Dev leverage this
information so that you do not need to resolve the same conflicts.

If you want DesignSync to ignore merge edges, specify -noedge:

  stcl> url resolveancestor -noedge -version Trunk top.v
  1.2

Version 1.2 is the branch point where Dev was branched from
Trunk. This version is the closest common ancestor if you do not
consider the merge of 1.5 and 1.2.1.3.
```

Example Showing Common Ancestor Using Branch and Version Arguments

```
In the previous examples, the argument to "url resolveancestor" was
a local file. You can also specify a version object:

  stcl> url resolveancestor -version Trunk [url vault top.v]1.2.1.7
  1.5

or a branch object:

  stcl> url resolveancestor -version Trunk [url vault top.v]Dev
  1.5

In the case of a branch, DesignSync uses the Latest version on
the specified branch.
```

url resolvetag

url resolvetag Command

NAME

```
url resolvetag      - Returns the version number associated with a
                      selector
```

DESCRIPTION

This command returns the version number to which a specified selector (typically a version or branch tag) or selector list resolves.
- If you specify a version selector (for example, a version tag), the corresponding version number is returned.
- If you specify a branch selector (for example, a branch tag), the version number of the Latest version on that branch is returned.
- If you do not specify a selector (no -version option), the version number of the Latest version on the current branch is returned.
- If the selector list does not resolve to a version, an exception is thrown.

Specify a versionable object as the argument to this command. If you specify any other object type, an empty string is returned.

SYNOPSIS

```
url resolvetag [-version <selector>[,<selector>...]] [--]
    <argument>
```

ARGUMENTS

- Module (Module-based)
- Module Member (Module-based)
- DesignSync Object (File-based)

Specifies one of the following arguments:

Module (Module-based)

<module>	Specifies the module for which you want the version number corresponding to the version option. If no version option is given, returns the version number of the currently fetched version for a workspace module, or the version associated with version 1:Latest for a server module.

Module Member (Module-based)

<module member>	Specifies the module member for which you want any associated tag. The command works correctly irrespective of whether the member is the Latest version on the members branch.

For example, for collection objects using the citags system, as part of the Custom Type System, the url resolvetag command returns any tags associated with the objects.

DesignSync Object (File-based)

<DesignSync object> Specifies the DesignSync object for which you want the version number of the current version.

OPTIONS

- -version
- --

-version

-version <selector> Specifies the selector list (typically a branch or version tag) for which you want the corresponding version number returned. The default behavior (if -version is not specified) is to return the version number of the Latest version on the current branch.

Note: To use -version to specify a branch, specify both the branch and version as follows: '<branchtag>:<versiontag>', for example, 'Rel2:Latest'. You can also use the shortcut, '<branchtag>:', for example "Rel2:". If you don't explicitly specify the branch selector in this way, DesignSync does not resolve the selector as a branch selector.

--

-- Indicates that the command should stop looking for command options. Use this option when the argument to the command begins with a hyphen (-).

RETURN VALUE

For all valid versionable objects, returns the version number of the current version of this object or the version number

```
corresponding to the selector specified with the -version option.

For any non-valid objects, returns an empty list.

For nonexistent object: raises an error.

Note: Use the -version argument to specify the selector of the
object.  If the version specified by -version does not exist, 'url
resolvetag' raises an error.
```

SEE ALSO

```
url tags
```

EXAMPLES

Example Showing a Resolved Version Tag

```
This example returns the version of "top.v" that is tagged "gold":
  stcl> url resolvetag -version gold top.v
  1.2.2.3
```

Example Showing the Latest Version of an Object

```
This example returns the version number of the Latest "top.v" on the
current branch, which in this case is different than the version
in the work area:
  stcl> url resolvetag top.v
  1.4
  stcl> url versionid top.v
  1.2
```

Example Showing the Latest Version on a Specified Branch

```
This example returns the Latest version of "top.v" on the branch
tagged "Rel2.1". Because "top.v" is not in the work area, you
must specify the vault.
  stcl> url resolvetag -version Rel2.1:Latest [url vault top.v]
  1.5.1.4
```

Example of Using a Selector List

```
This example specifies a selector list. The file "samp.asm" does
not have a version tagged "beta", so the version corresponding to
the tag "alpha" is returned. The file "top.v" does not have a
version corresponding to "alpha" or "beta" version tags, so an
exception is thrown.
  stcl> url resolvetag -version beta,alpha samp.asm
  1.3
  stcl> url resolvetag -version beta,alpha top.v
  som-E-152: No Such Version.
```

Example Showing a Non-Existent Module Version (Module-based)

```
In the following example, the 'url resolvetag' command is run on a
module version that does not exist. The -version option is used to
specify the module version.
  stclc> url resolvetag -version GOLDer Cpu%0
  SomAPI-E-101: No Such Version.
```

url retired

url retired Command

NAME

```
url retired         - Returns whether a branch is retired
```

DESCRIPTION

```
This command checks whether an object's branch is retired
(returns 1) or not (returns 0).

The object can be:
  o A versionable object (file or collection), in which case the
    retired status of the current branch is returned.
  o A branch, in which case the retired status of that branch is
    returned.
  o A vault, in which case the retired status of branch 1 is
    returned.

The command returns "0" for all other object types.
```

SYNOPSIS

```
url retired [--] <argument>
```

ARGUMENTS

- DesignSync Object

```
Specifies one of the following arguments:
```

DesignSync Object

```
<DesignSync object> Specifies the branch of a DesignSync object
                    that you want to know is retired or not.
                    Returns 1 if the specified branch is retired,
                    otherwise 0.
```

OPTIONS

- --

--

```
--                  Indicates that the command should stop looking for
                    command options. Use this option when the argument
                    to the command begins with a hyphen (-).
```

RETURN VALUE

```
For any valid object, returns 1 (Tcl TRUE) if it is retired,
otherwise returns 0 (Tcl FALSE).

For invalid or non-existent objects, returns 0.
```

SEE ALSO

```
retire, vhistory, url exists, url registered
```

EXAMPLES

Example Showing the Status of a File Before/After Retirement

```
This example checks the retired status of a file's current branch
before and after retiring the branch.
  dss> url retired top.gv
  0
  dss> retire -keep top.gv
  Beginning Retire operation...

  top.gv: Branch 1 Retired in the vault.

  Retire operation finished.
  dss> url retired top.gv
  1
  dss> retire -unretire top.gv
  Beginning Retire operation...

  top.gv: Branch 1 in Vault is now active.

  Retire operation finished.
  dss> url retired top.gv
  0
```

Example Showing the Status of a Retired Branch

```
This example checks the retired status of the "Rel2.1" branch of
"top.v" by specifying the branch object itself:

  dss> url retired "sync://host:3002/Projects/Mod1/top/top.gv;Rel2.1"
  1

or from stcl/stclc, you can use "url vault" to simplify the
specification of the branch URL:

  stcl> url retired [url vault top.gv]Rel2.1
  1
```

url rmprop

url rmprop Command

NAME url rmprop

```
url rmprop          - Removes specified properties for a module
                      object from the local metadata
```

DESCRIPTION

```
This command removes the specified properties from the local metadata
for the specified module object.  It is used to remove the properties
that indicate that an object was renamed or removed on the branch
merged into the workspace.
```

SYNOPSIS

```
url rmprop <argument> <property>
```

ARGUMENTS

- Workspace Module Member

Workspace Module Member

```
<workspace          Specify an object in the workspace to remove
module member>      the specified property from.
```

OPTIONS

- Property

Property

```
<property>          Specify the property to remove from the specified
                    object. The following properties can be removed
                    from an object.:

                    ci_rename - indicates that the object was renamed
                                on the branch merged into the
                                workspace. Removing this property
                                removes that information.

                    ci_remove - indicates that the object was removed
                                on the branch merged into the
                                workspace. Removing this property
                                cancels that action.
```

RETURN VALUE

```
This command returns a TCL value of null ("").
```

```
For workspace module members (<path/>File1.txt): returns a success
message indicating that the property has been deleted.

For other objects: Not applicable; raises error.

Note: The command returns a success message even if the property
being removed was not set for the object.
```

SEE ALSO

```
populate, url getprop, url setprop
```

EXAMPLES

Example Showing Removal of the ci_rename Property

```
This example shows removing the ci_rename property from the rom.h
file in the current directory.

dss>  pwd
 /home/rsmith/MyModules/rom

dss> url rmprop rom.h ci_rename
 Success: deleted property ci_rename
```

Example Showing Removal of the ci_remove Property

```
This example shows removing the ci_remove property from the rom.h
file using the full path to the module object.

dss> url rmprop /home/rsmith/MyModules/rom/rom.h ci_remove
 Success: deleted property ci_remove
```

url root

url root Command

NAME

```
url root            - Returns the workspace root for a given path
```

DESCRIPTION

```
This command returns the workspace root for a given path,
which is either the given path or a parent directory. If there
has been no "setroot" command performed, an empty string ("") is
returned.
```

SYNOPSIS

```
url root [--] <path>
```

ARGUMENTS

- Path

Path

```
<path>              Specify a local directory path or module instance
                    name.
```

OPTIONS

- --

--

```
--                  Indicates that the command should stop looking for
                    command options. Use this option when the object
                    you specify begins with a hyphen (-).
```

RETURN VALUE

- Return Values for Modules (Module-based)
- Return Values for Files (File-based)

Return Values for Modules (Module-based)

```
For a module instance name, returns the directory path to the workspace
root directory.

For any object within a root structure, returns the directory
path to the workspace root directory.
```

```
For all other values, returns nothing indicating that there is no
associated workspace root directory.
```

Return Values for Files (File-based)

```
For any object within a root structure, returns the directory
path to the workspace root directory.

For all other values, returns nothing indicating that there is no
associated workspace root directory.
```

SEE ALSO

```
setroot, mkmod, add
```

EXAMPLES

- Viewing the Root Directory for a Module Workspace (Module-based)
- Viewing the Root Directory for a File-Based Workspace (File-based)

Viewing the Root Directory for a Module Workspace (Module-based)

```
This example shows the root directory for the rom.doc file in
the Doc subdirectory of the ROM module.
Note: This example uses the absolute path to show the directory
structure, but the command also accepts relative paths.

dss>  url root /home/rsmith/MyModules/ROM/Doc/rom.doc
  /home/rsmith/MyModules
```

Viewing the Root Directory for a File-Based Workspace (File-based)

```
This examples shows the oort directory for the chip.doc file in the
files-based Chip project.

Note: This example uses the absolute path to show the directory
structure, but the command also accepts relative paths.

dss> url root /home/rsmith/projects/chip/chip.doc
 /home/rsmith/projects
```

url selector

url selector Command

NAME

```
url selector        - Returns an object's persistent selector list
```

DESCRIPTION

```
This command returns the persistent selector list (comma-separated
list of selectors stored in an object's local metadata) associated
with a specified object. You can specify a versionable object (file
or collection object) or a local folder. If you specify any other
object type, or a nonexistent object, an exception is thrown.

The selector list returned by 'url selector' is determined as
follows:
  1. If the object has its own persistent selector list,
     return that selector list.
  2. Otherwise, return the persistent selector list of the first
     folder from the parent folder to the file-system root (/)
     that has a persistent selector list. In other words, the object
     inherits its parent folder's selector list.
  3. Otherwise, return "Trunk", which is the default persistent
     selector list.

Some revision-control commands use the persistent selector list to
determine which branch or version to operate on, unless overridden
by an explicit -version or -branch option. See the "selectors" help
topic for more information on selectors and selector lists.

Note that the "P" data key for the 'ls' command and the Selector
column of the List View (in the DesignSync graphical user
interface) also report an object's persistent selector list.
```

SYNOPSIS

```
url selector [--] <argument>
```

ARGUMENTS

- Workspace Module (Module-based)
- DesignSync Object (File-based)

```
Specify one of the following arguments:
```

Workspace Module (Module-based)

```
<workspace module>  Specifies the top level workspace module for
                    which you want the associated persistent
                    selector list.
                    You also get the selectors inherited from the
                    href of sub-modules.
```

DesignSync Object (File-based)

```
<DesignSync object> Specifies the DesignSync object for which you
                    want the associated persistent selector list.
```

OPTIONS

- --

--

```
--                  Indicates that the command should stop looking
                    for command options. Use this option when the
                    object you specify begins with a hyphen (-).
```

RETURN VALUE

```
For all valid objects, returns the persistent selector list set for
the object; if no persistent selector list is set, returns the first
persistent selector list found from the parent folder upward to the
file system root (/). If no persistent selector list is inherited,
returns the default persistent selector list, Trunk.

For any invalid or non-existant objects, returns an applicable
error.
```

SEE ALSO

```
setselector, selectors, ls
```

EXAMPLES

- Example Showing the Persistent Selector for the Module (Module-based)
- Example Showing the Persistent Selector List for a File (File-based)
- Example Showing a Single Persistent Selector (File-based)

Example Showing the Persistent Selector for the Module (Module-based)

```
This example uses 'url selector' command to find the persistent
selector for module1:
 stcl> url selector Module1%0
 Trunk:
```

Example Showing the Persistent Selector List for a File (File-based)

```
In the following example, a local work area folder called folder1
contains a file called top.v. The recursive 'setselector' command
sets the persistent selector list for the folder and its contents.
   dss> setselector -recursive Gold,Silver,Bronze folder1
   dss> cd folder1
   dss> url selector .
   Gold,Silver,Bronze
   dss> url selector top.v
   Gold,Silver,Bronze
```

Example Showing a Single Persistent Selector (File-based)

```
In the following example, you want to populate an empty work area
called projectA with the latest versions of objects that have a
"Gold" branch or specific versions tagged "Gold".
   dss> cd projectA
   dss> setselector Gold .
   dss> populate
   <Files are fetched into the work area, including one called gc7.v>
   dss> url selector gc7.v
   Gold

You now fetch a different version of gc7.v. The selector list for
gc7.v is updated while the selector list for the folder remains "Gold".

   dss> co -version rel2.1 gc7.v

   Beginning Check out operation...

   Checking out: gc7.v          : Success - Fetched  version: 1.6

   Checkout operation finished.

   dss> url selector gc7.v
   rel2.1
   dss> url selector .
   Gold
```

url servers

url servers Command

NAME

```
url servers         - Returns server-list definitions
```

DESCRIPTION

```
This command returns the SyncServers or vaults specified
in the server-list (sync_servers.txt) files.

Using server-list files simplifies the selection of SyncServers or
vaults from DesignSync (from the Workspace Wizard) and DesignSync
DFII (from the Vault Browser). When setting up a work area,
users can select a SyncServer or vault
using a friendly name instead of specifying the URL.

The user server-list file is:

  <SYNC_USER_CFGDIR>/sync_servers.txt

where SYNC_USER_CFGDIR is the environment variable that specifies
your directory for user-specific customization files.
If you have not defined the SYNC_USER_CFGDIR environment variable,
then DesignSync looks for:

  <home>/.synchronicity/sync_servers.txt

where <home> is your home directory as defined by $HOME on UNIX
or your user profile, which is managed by the User Manager tool,
on Windows platforms.

The site server-list file is:

  <SYNC_SITE_CNFG_DIR>/sync_servers.txt

where SYNC_SITE_CNFG_DIR is the environment variable that specifies
the directory for site-specific customization files. If
you have not defined the SYNC_SITE_CNFG_DIR environment variable,
then DesignSync looks for:

  <SYNC_DIR>/custom/site/config/sync_servers.txt

The user and site sync_servers.txt files are not provided as part
of the Synchronicity installation, so you need to create them if
they do not already exist.

The enterprise server-list file is:

  <SYNC_ENT_CUSTOM>/config/sync_servers.txt

where SYNC_ENT_CUSTOM is the environment variable that specifies
```

```
the directory for enterprise-specific customization files. If
you have not defined the SYNC_ENT_CUSTOM environment variable,
then DesignSync looks for:

  <SYNC_DIR>/custom/enterprise/config/sync_servers.txt

The user, site, and enterprise sync_servers.txt files are not
provided as part of the Synchronicity installation, so you need to
create them if they do not already exist.

The syntax for the server-list file is:

  NAME <name>         Friendly name for the SyncServer or vault
  REFERENCE <url>     The complete URL to the SyncServer or vault
  DESCRIPTION <text>  Brief description of the SyncServer or vault

The following rules apply to the sync_servers.txt files:
  o The NAME keyword begins a new SyncServer or vault definition. It
    must appear before the REFERENCE and DESCRIPTION keywords.
  o The REFERENCE and DESCRIPTION keywords can appear in any order.
  o The keywords are case insensitive.
  o Keywords must be the first non-whitespace characters on a line.
  o The DESCRIPTION field is optional.
  o The DESCRIPTION text can span multiple lines and is terminated
    by a blank line or a keyword as the first non-whitespace
    characters on a line. A comment itself can therefore not
    include a blank line; otherwise, the remaining comment will
    be ignored.
  o Each NAME value must be unique; duplicates are ignored.
  o If the same NAME value appears in both the user and site
    files, the user definition takes precedence.
    The order of precedence is:
   1) user
   2) site
   3) enterprise
  o Comments are indicated by a pound sign (#) as the first
    non-whitespace character on a line.
      Note: Text that does not follow a keyword is ignored and
      therefore behaves like a comment. However, because the
      supported keywords may change in future releases, precede all
      comments with #.
```

SYNOPSIS

```
url servers [-all | -enterprise | -site | -urls* | -user]
```

OPTIONS

- -all
- -enterprise
- -site

- -urls
- -user

-all

```
-all        Returns both site and user server lists, with
            duplicates removed (user definitions have
            precedence over site definitions).
```

-enterprise

```
-enterprise Returns only the enterprise server list.
```

-site

```
-site       Returns only the site server list.
```

-urls

```
-urls       Preserves the previous behavior of returning only the
            REFERENCE URL for each SyncServer or vault with a
            unique NAME from both the site and user
            sync_servers.txt files.

            *Note: This option is the default behavior in order to
            maintain backward compatibility. The -urls option
            will be removed in a future DesignSync release, and the
            default behavior (no options specified) will
            change. Therefore, it is recommended that you use the
            -enterprise, -site, -user, or -all option.
```

-user

```
-user       Returns only the user server list.
```

RETURN VALUE

```
If -url or no option is specified, a list of REFERENCE URLs:
  url url ...

Otherwise, a list of lists, with each sublist containing the NAME,
REFERENCE, and DESCRIPTION values:
```

```
  {{name} {url} {description}} {{name} {url} {description}} ...
```

EXAMPLES

```
A site sync_servers.txt file contains the following:

  # This sync_servers.txt file is used by the entire
  # Marlboro site.
  NAME Doc Vault
  REFERENCE sync://docserver:2647/Projects/docs
  DESCRIPTION Server for documentation source files.
  Only the documentation group can lock files.

  NAME Source
  REFERENCE sync://src:3001

and a user sync_servers.txt file contains the following:

  NAME My Server
  REFERENCE sync://localhost:2647

  NAME Source
  REFERENCE sync://src.myco.com:3001
  DESCRIPTION The company-wide source repository

The following are the results of 'url servers'. Note that in the
combined list (with -all specified), 'url servers' returns the
user's "Source" definition, because the user's sync_servers.txt
file takes precedence over the site file.

stcl> url servers -user
{{My Server} {sync://localhost:2647} {}}
{{Source} {sync://src.myco.com:3001} {The company-wide source
           repository.}}
stcl> url servers -site
{{Doc Vault} {sync://docserver:2647/Projects/docs}
{Server for documentation source files. Only the documentation group
can lock files.}} {{Source} {sync://src:3001} {}}
stcl> url servers -all {{My Server} {sync://localhost:2647} {}}
{{Source} {sync://src.myco.com:3001} {The company-wide source
           repository.}}
{{Doc Vault} {sync://docserver:2647/Projects/docs}
{Server for documentation source files. Only the documentation group
can lock files.}}
```

url setprop

url setprop Command

NAME

```
url setprop         - Sets a property on an object
```

DESCRIPTION

This command lets you set the value of a property on most objects (on notes you can change existing properties, but not add new ones). Properties are specified as a name (typically a short identifier) and a value, which can be a string of any length. Such properties are stored with the metadata representing the object.

IMPORTANT: The property prefix "Sync" is reserved for DesignSync properties. You should not create any properties that begin with this reserved prefix. While this prefix is case sensitive, DesignSync recommends, to minimize confusion, that you avoid using "sync" with any casing variant as a prefix to any custom properties.

For note URLs, both the object and property must exist and the property value supplied must be legal for its property type. The new property value specified in this command is checked against the current value of the property. If they are the same, no change is made to the object.

Note that the "special" properties that are supported by the url properties command are not available to url setprop. For example, if url properties reports that an object is locked by someone, you cannot unlock it with url setprop by passing in "locked 0".

You can use url setprop with most object types. However, depending on the type, the command may only work in server-side scripts, such as when accessing notes. User-defined properties are not supported for configurations.

Because DesignSync automatically determines the datatype of the vault, it may assign a datatype that you do not want. For example, it may assign the binary datatype to an ASCII file. In such cases, you can use the 'url setprop' command to change the vault datatype.

The successful execution of this command on a note object causes an atomic note modify event and fires the corresponding triggers in response. If the property value equals the current value of the property, no event is generated.

Note: If you need to set more than one property on the same note, it is preferable to use the note setprops command, because it is more efficient and reduces trigger activity.

You can use the "url setprop" command to change the checkin comments of objects checked into a vault.

Note: This command does not change the comments associated with tags. To change tag comments remove the tag and add it back again.

The "url setprop" command is subject to access controls on the server. For more information, see the ENOVIA Synchronicity Access

```
Control Guide.
```

SYNOPSIS

```
url setprop [--] <Object_url> <prop_name> <prop_value>
```

OPTIONS

- --

--

```
  --            Indicates that the command should stop looking for
                command options. Use this option when property
                names or values begin with a hyphen (-).
```

OPERANDS

- Object URL
- Property Name
- Property Value

Object URL

```
<Object_url>  A valid object URL.
```

Property Name

```
<prop_name>   The name of the property to set on the object.

              You can specify the special property name DataType to
              assign the data type of the vault.
```

Property Value

```
<prop_value>  The value of the property to set on the object.

              When you use the special DataType property for a vault,
              it can take one of the following values:

               o ascii or text - changes the vault data type ASCII.
               o binary        - changes the vault data type to
```

```
                    binary.
          o undefined     - lets DesignSync determine the vault
                            data type at the next check in,
                            based on the file's contents.
```

RETURN VALUE

```
For all valid objects, returns the value set for the new property.

For all invalid or non-existent objects, returns an error.
```

SEE ALSO

```
note getprop, note setprops, url getprop, url properties
```

EXAMPLES

- Example of Setting a User-Defined Property on a Module Workspace (Module-based)
- Example of Setting a User-Defined Property on a Module Member (Module-based)
- Example of Setting a User-Defined Property on a Server Vault (File-based)
- Example of Changing the DataType value for an Object (File-based)
- Example of Changing a Comment (File-based)

Example of Setting a User-Defined Property on a Module Workspace (Module-based)

```
The example uses the 'url setprop' command to change the user defined
property "respuser" for a module workspace.

stcl> url getprop Chip%0 respuser
tadams

stcl> url setprop Chip%0 respuser rsmith
rsmith
```

Example of Setting a User-Defined Property on a Module Member (Module-based)

```
This example uses the 'url setprop' command to change the user defined
property "respuser" for module member File1.txt:
 stcl> url getprop [url vault File1.txt] respuser
 tadams

 stcl>  url setprop [url vault File1.txt] respuser rmsith
 rsmith

 stcl> url getprop [url vault File1.txt] respuser
```

```
  rsmith
```

Example of Setting a User-Defined Property on a Server Vault (File-based)

```
This server-side example sets and displays a CcList user-defined
property on a vault.
  stcl> url setprop "sync:///Projects/myproj/foo.v;" CcList {sal jason mark}
  stcl> puts [url getprop "sync:///Projects/myproj/foo.v;" CcList]
```

Example of Changing the DataType value for an Object (File-based)

```
This example changes the vault data type to ascii. "url vault" is
used to identify the vault file associated with the workspace file.
  stcl> url setprop [url vault samp.asm] DataType ascii
```

Example of Changing a Comment (File-based)

```
This example changes the comment stored with the object version.
stcl> url setpop "[url vault samp.asm];1.3" log "SD 1550 - changed
comment to include SD number for correction."

Note: If the log property value is not specified on the command,
DesignSync prompts for an interactive comment specified either on the
command line or by spawning the defined file editor. When the file
editor is launched to edit the comments, it is initiallized with the
current value of the log property. For more information on defining a
file editor, see the DesignSync Data Manager Administrator's Guide,
"General Options."
```

url syslock

url syslock Command

NAME

```
url syslock         - Sets a system lock on a lock name or file path
```

DESCRIPTION

```
This command lets you manipulate arbitrary system locks by
name. The name may be a logical lock name or a path to an actual
file. Note that the locks are advisory locks. This means that
```

there is nothing from preventing a user from performing an action for which another user has obtained a lock. It is up to each user to check for the existence of the lock before performing the operation. This process is cooperative.

There are two types of locks and three basic locking operations. Locks may be shared or exclusive. A shared lock (also called a read lock) may be held by several processes at one time. An exclusive lock (also called a write lock) may be held by only one process at a time. Furthermore, an exclusive lock will not be granted if any shared lock is in place.

The three locking operations are acquire, yield, and release. Acquire and release simply obtain and give up the named lock. The yield call allows the holder of a lock to release that lock temporarily and then re-obtain it. In this way processes that expect to hold a lock for a long period of time may choose to release the lock temporarily to allow other processes to obtain the lock, perform some relatively fast operation, and then release the lock again. At this point the original process may again acquire the lock. Normally, acquire requests are counted and the lock released only when the reference count drops to zero. The -all option overrides this behavior and forces a release. The original reference count is then restored when the lock is re-acquired. This option is only recognized by the yield call.

A timeout may be specified when calling to acquire or yield a lock. By default the call blocks until the specified lock becomes available. If the timeout value is supplied, however, this will be taken as the maximum number of seconds to wait for the specified lock to become available. Should the timeout period expire, an error is returned.

The canonization flag specifies that the string supplied should be interpreted as a file path name and, further, that the name should be resolved into a canonical path. The canonization process will take file system mount points into consideration. For example, if your home directory is mounted to /u/home/user on system_X and to /home1/user on system_Y, and your home directory actually resides at /users/home on system_A, then the canonization process results in the path system_A:/users/home regardless of the system from which you invoke the lock request.

The realmount and realpath calls canonize the supplied paths. For realmount, the path is resolved down to the mount point. For realpath, the path is only resolved down to the root of the local file system; no mount point resolution is taken into account.

The showlocks call displays the locks that the current process holds. The output format includes the lock name, its index within the master lock file (the file used to acquire operating system level locks), the number of read locks pending, and the number of write locks pending.

SYNOPSIS

```
url syslock -acquire       <name_or_path> [-canonize] [-shared]
                           [-timeout secs]
url syslock -yield         <name_or_path> [-canonize] [-shared]
                           [-timeout secs] [-all]
url syslock -release       <name_or_path> [-canonize]
url syslock -realmount     <name_or_path>
url syslock -realpath      <name_or_path>
url syslock -showlocks

Note: The <name_or_path> argument must follow the
      -acquire/-yield/-release/-realmount/-realpath option.
```

OPTIONS

-all

```
-all             Only available with -yield. The default behavior
                 is that a lock is released only when the
                 reference count drops to zero. The -all option
                 overrides this behavior and forces a release. The
                 original reference count is then restored when
                 the lock is re-acquired.
```

-acquire

```
-acquire         Obtains the specified lock.
```

-canonize

```
-canonize        Specifies that the string supplied should be
                 interpreted as a file path name and that the name
```

```
                  should be resolved into a canonical path. The
                  canonization process takes file system mount
                  points into consideration. For example,
                  if your home directory is mounted to /u/home/user
                  on system_X and to /home1/user on system_Y, and
                  your home directory actually resides at
                  /users/home on system_A, then the canonization
                  process results in the path system_A:/users/home
                  regardless of the system from which you invoke
                  the lock request.
```

-realmount

```
 -realmount       Canonizes the specified path, resolved down to
                  the mount point.
```

-realpath

```
 -realpath        Canonizes the specified path, resolved down to the
                  root of the local file system; no mount point
                  resolution is taken into account.
```

-release

```
 -release         Gives up the specified lock.
```

-shared

```
 -shared          Specifies that the lock to be acquired or yielded
                  is a shared (read) lock. When -share is not
                  specified, the lock is an exclusive (write) lock.
```

-showlocks

```
 -showlocks       Displays the locks that the current process holds.
```

-timeout

```
 -timeout <secs>  Available with -acquire and -yield. By default, the
                  call blocks until the specified lock becomes
                  available. If a timeout value is supplied, you
                  specify the maximum number of seconds to wait for
```

the specified lock to become available. Should the timeout period expire, an error is returned.

-yield

-yield	Allows the holder of a lock to release that lock temporarily and then re-obtain it.

--

--	Indicates that the command should stop looking for command options. Use this option when the argument to the command begins with a hyphen (-).

RETURN VALUE

For a logical lock name (Asic/x.v:lnx:16738,rd:0,wr:1): Performs the action specified by the options, but provides no return value.

Note: The logical lock name format includes the lock name, its index within the master lock file (the file used to acquire operating system level locks), the number of read locks pending, and the number of write locks pending.

For other client-side objects: Performs the action specified by the options, but provides no return value; 'url syslock' does not verify that the object exists.

For server-side objects: Raises error.

EXAMPLES

Tihs example shows acquiring a lock, viewing the lock, and then releasing the lock.

```
stcl> url syslock -acquire /home/users/dave/sample.txt -canonize
Acquired: /home/users/dave/sample.txt
stcl> url syslock -showlocks
Current Locks: /home/users/dave/sample.txt:lnx:16738,rd:0,wr:1
stcl> url syslock -release sample.txt -canonize
Released: /home/users/dave/sample.txt
```

url tags

url tags Command

NAME

```
url tags            - Returns the version tags associated with
                      a specified object
```

DESCRIPTION

```
This command returns the list of version tags associated with the
specified object. Tags are listed with "Latest" first, if
applicable, then in order from oldest to newest. If the object has
no associated version tags, a null string is returned.

By default the command returns the  version tags associated with
the currently fetched version of the object. Use the -version
option to fetch the tags associated with a particular version
of the object.

Note: You can use the -btags argument with the url tags command to
specify the branch tags, rather than the version tags.
```

SYNOPSIS

```
url tags [-btags] [-version <selector>] [--] <argument>
```

ARGUMENTS

- Module (Module-based)
- Module Member (Module-based)
- DesignSync Object (File-based)

```
Specifies one of the following arguments:
```

Module (Module-based)

```
<module>            Specifies the module for which you want the
                    associated version tags.
```

Module Member (Module-based)

```
<module member>     This is not a valid argument type.
                    Returns an empty string "".
```

DesignSync Object (File-based)

```
<DesignSync object> Specifies the DesignSync object for which you
                    you want the associated version tag.
```

OPTIONS

-btags

```
-btags              Specifies displaying branch tags instead of the
                    version tags for the specified argument.
```

-version

```
-version <selector>  Specifies the version of the local object
                     (file or collection) for which you want the
                     associated tags.
                     The  -version option is ignored if you specify
                     a version or branch as the argument to
                     "url tags". See the "selectors" help topic for
                     details on selectors.

                     Note: To use -version to specify a branch,
                     specify both the branch and version as follows:
                     '<branchtag>:<versiontag>', for example,
                     'Rel2:Latest'. You can also use the shortcut,
                     '<branchtag>:', for example "Rel2:".  If
                     you don't explicitly specify the branch
                     selector in this way, DesignSync does not
                     resolve the selector as a branch selector.
```

--

```
--                  Indicates that the command should stop looking
                    for command options. Use this option when the
                    argument to the command begins with a hyphen
                    (-).
```

RETURN VALUE

```
For all valid objects, returns the specified list of tags associated
with the specified object, with "Latest" first, if applicable, then
in order from oldest to newest.  If no tags exit, it returns an empty
list.

For other invalid or non-existent objects, returns appropriate error.
```

SEE ALSO

```
url resolvetag, url versionid, tag
```

EXAMPLES

Example Showing the Tags Associated with a Module (Module-based)

```
This example uses 'url tags' to get the tag associated with a
module:
 stcl> url tags Module1%0
 Latest
```

Example Showing the Tags Associated with a File (File-based)

```
This example returns the version tags associated with the
current version of "top.v". Because the current version is also the
latest version on that branch, DesignSync automatically associates
"Latest" with that version.

  dss> url tags top.v
  Latest Gold
```

Example Showing the Tags Associated with a Specified File Version (File-based)

```
This example returns the tags associated with version 1.2 of "top.v":
  dss> url tags -version 1.2 top.v
  Bronze

You can also specify the version object itself. You might do this
if you do not have the object in your work area or if you are using
"url tags" in a server-side script. Specifying the -version option
would have no effect in this case because you have specified a
version as the command argument:
```

```
stcl> url tags [url vault top.v]1.2
Bronze
```

url users

url users Command

NAME

```
url users           - Returns all users defined for an object's
                      server
```

DESCRIPTION

```
This command returns the list of all users defined for a server.
If no users exist, an empty string is returned.

The <object> parameter is optional and is always ignored. This
parameter is retained for backward compatibility. If you supply
an object URL, the URL must be valid.

Note that the object can be the server root itself, such as:
  url users sync:///

Note: 'url users' is a server-side only command. For more
information, type 'help server-side'.
```

SYNOPSIS

```
url users <path>
```

OPTIONS

```
none
```

OPERANDS

- Path to the Server

Path to the Server

```
<path>            The path to the server. Optional.
```

RETURN VALUE

```
For a server or server-side object, returns a list of
the users defined for the server.

For any invalid objects or non-existent objects, returns an error.
```

SEE ALSO

```
url contents, url projects, server-side, rstcl
```

EXAMPLES

```
This example returns the users currently defined for the Sportster
project. Because users are defined on a per-server basis, all users
defined for the holzt:2647 server are returned.

1. In <SYNC_DIR>/custom/site/share/tcl, create 'users.tcl' that
   contains the following lines:
     puts [url users sync:///Projects/Sportster]

2. From your browser, issue the following URL:
      http://holzt:2647/scripts/isynch.dll?panel=TclScript&
      file=users.tcl

   The browser displays the users currently defined for the
   Sportster project:
       sync:///Users/goss
       sync:///Users/barbg

   You could also execute this script from a DesignSync client
   using the rstcl command.
```

url vault

url vault Command

NAME

```
url vault           - Returns the URL of an object's vault
```

DESCRIPTION

- Note for modules (Module-based)

```
This command returns the URL of the vault object associated with
the specified object. If the object is a directory, the
vault-side directory that it is associated with is returned.

The general syntax for the url commands is described under
"help url".
```

Note for modules (Module-based)

```
This is an internal command, not very useful except
when a DesignSync vault is upgraded to a module and some DesignSync
objects have versions which are not present in any module version.
In this case, you need to know the vault URL of the object in order
to retrieve properties from that member version.
```

SYNOPSIS

```
url vault [-modulecontext <context>] <argument>
```

ARGUMENTS

- Workspace Module (Module-based)
- Module Folder (Module-based)
- Module Member (Module-based)
- Designsync Object (File-based)

```
Specify one or more of the following arguments:
```

Workspace Module (Module-based)

```
<workspace module>  Specifies the workspace module for which you
                    want the  URL of the vault.
```

Module Folder (Module-based)

```
<module folder>     This is not a valid argument type.
                    Returns the following error as module folders
                    do not have a direct vault association:
                    Error: <module folder>: This folder is a module
                    member and does not have an individual vault
```

value.

Module Member (Module-based)

<module member>	Specifies the module member for which you want the URL of the vault. Note: Module members are operated on in the context of a module and have a specific vault URL. However, the same module member in different versions of the module can have different vault URL. This is because of the way module members are renamed. See the mvmember command for more information.

Designsync Object (File-based)

<DesignSync object>	Specifies the DesignSync object for which you want the URL of the vault.

OPTIONS

- -modulecontext (Module-based)

-modulecontext (Module-based)

-modulecontext <context>	Specifies a module context, allowing the retrieval of a vault for a module member that is not in the workspace. In this case, the argument must be the natural path of the object.

RETURN VALUE

For all valid objects, returns the URL of the vault associated with the specified object.

For any non-valid objects or non-existent objects, returns an empty list.

SEE ALSO

url, setvault

EXAMPLES

Example of Getting the Module Vault Information (Module-based)

```
This example uses the 'url vault' command to get the url
of a module's vault.
  stcl> url vault Module1%0
  sync://srv2.ABCo.com:2647/Modules/Module1
```

Example of Getting Vault for the Current Working Directory (File-based)

```
This example returns the vault folder for the current directory using
relative and absolute paths:
  dss> url vault .
  sync://holzt:2647/Projects/Sportster

  dss> url vault /home/goss/Projects/Sportster
  sync://holzt:2647/Projects/Sportster
```

Example of Getting the Vault for a Specified File (File-based)

```
This example returns the vault for top.v:
  dss> url vault top.v
  sync://holzt:2647/Projects/Sportster/top/top.v;
```

Example of Using the url vault command within a Command (File-based)

```
This example uses 'url vault' to change directory into a vault
folder:
  stcl> scd [url vault top.v]
  stcl> spwd
  sync://holzt:2647/Projects/Sportster/top/top.v;
```

url versionid

url versionid Command

NAME

```
url versionid       - Returns the version number of the specified
                      object
```

DESCRIPTION

- Module Notes (Module-based)

```
This command returns the version number of the specified object.

- For managed objects, the current version number (as stored in the
  local metadata) is returned.
- If the object is a reference, the version number is preceded by
  "Refers to:".
- If the object is locked, the current version number and upcoming
  version number are returned.
- For version objects specified with a version number, that
  version number is returned.

Notes:
o  To get the version number of a selector list or a tag, use url
   resolvetag command.

o The return value of 'url versionid' is the same as the version
  returned by the 'ls -report R' command and the Version column of
  the List View (from the DesignSync graphical user interface).
```

Module Notes (Module-based)

```
o You cannot specify a version number for a module or module member
  object.

o Module members that have been added but not checked in are
  considered "unmanaged."
```

SYNOPSIS

```
url versionid [--] <argument>
```

ARGUMENTS

- Workspace Module (Module-based)
- Module Member (Module-based)
- DesignSync Object (File-based)

Specifies one or more of the following arguments:

Workspace Module (Module-based)

<workspace module> Specifies the module for which you want the current version number.

Module Member (Module-based)

<module member> Specifies the module member for which you want the current version number.

DesignSync Object (File-based)

<DesignSync objects> Specifies the DesignSync object for which you want the current version number.

OPTIONS

- --

--

-- Indicates that the command should stop looking for command options. Use this option when the argument to the command begins with a hyphen (-).

RETURN VALUE

For all valid objects, returns the version number of the object (1.3). If the object is a reference, the version number is preceded by "Refers to:". If the object is locked, the current version number and upcoming version number are returned (1.3 -> 1.4).

For invalid revision controlled objects, returns an empty list.

For any objects not under revision control, returns "unmanaged."

For any objects that do not exist, returns an appropriate error.

SEE ALSO

```
url versions, url branchid, url resolvetag, ls
```

EXAMPLES

Example Showing Different Return Values for Module Objects (Module-based)

```
The following example uses 'url versionid' to get the version
number of a module:
  stcl> url versionid Module1%0
  1.2

The following example uses 'url versionid' to get the version
number of a module member that is added but not yet checked in:
  stcl> url versionid File3.txt
  Unmanaged
```

Example Showing a Variety of Different Return Values for File Objects (File-based)

```
The following examples show the different return values for
the 'url versionid' command.

  stcl> ls -report R samp.asm samp.lst test.mem test.asm test.s19
  Version          Name
  -------          ----
  1.2              samp.asm
  1.1 -> 1.2       samp.lst
  1.2.1.2          test.mem
  Refers to: 1.1   test.asm
  Unmanaged        test.s19

For local managed objects, returns the current version number:
  stcl> url versionid samp.asm
  1.2
  stcl> url versionid test.mem
  1.2.1.2

For references, precedes the version number with "Refers to:":
  stcl> url versionid test.asm
  Refers to: 1.1

For locked objects, returns the current and next  version number:
  stcl> url versionid samp.lst
  1.1 -> 1.2

For unmanaged objects, returns "Unmanaged":
  stcl> url versionid test.s19
  Unmanaged
```

```
For version objects, returns the version number:
  stcl> url versionid [url vault samp.asm]1.1
  1.1

For folders, vaults, and other objects that do not have versions,
returns an empty string:
  stcl> url versionid .
  stcl> url versionid sync://myhost:2647
  stcl> url versionid [url vault samp.asm]

If the object does not exist, an exception is thrown:
  stcl> url versionid nosuchobject
  somapi-E: Unable to locate: nosuchobject
```

url versions

url versions Command

NAME

```
url versions        - Returns the URLs of an object's versions
```

DESCRIPTION

```
This command returns a list of URLs of the version objects
associated with the specified object. If the object is
a vault, the versions on the main branch (branch 1) are returned.
If the object is a branch, the versions on that branch are
returned. If the object is a local managed object, the versions
for that object's current branch are returned. If the object is
not a vault, branch, or managed object, then an empty list is
returned.

Note: In stcl/stclc mode, when specifying a URL that contains a
semicolon (;), surround the URL with double quotes.
```

SYNOPSIS

```
url versions <argument>
```

ARGUMENTS

- Module Branch (Module-based)
- Module Member (Module-based)
- Server Module (Module-based)

- DesignSync Object (File-based)

Specifies one or more of the following:

Module Branch (Module-based)

<module branch>	Returns the versions of the specified module branch. If it is a workspace module, applies the command to the current branch of that module.

Module Member (Module-based)

<module member>	Returns all versions associated with the module member or module member branch.

Server Module (Module-based)

<server module>	Returns all versions on all branches for the server module.

DesignSync Object (File-based)

<DesignSync object>	Specifies the DesignSync object for which you want the URLs of the associated versions.

OPTIONS

none

RETURN VALUE

For valid objects, returns a list of the version URLs associated with the object; either on the specified branch or all branches.

For any invalid object, returns empty list.

For nonexistent objects, raises error.

SEE ALSO

```
url contents
```

EXAMPLES

- Example of Getting Versions Associated with a Server Module (Module-based)
- Example Showing Versions on the Trunk Branch (File-based)
- Example Showing Versions on File Vault Object (File-based)

Example of Getting Versions Associated with a Server Module (Module-based)

```
This example uses 'url versions' command to get the list versions
for a server module:
  stcl> url versions sync://srv2.ABCo.com:2647/Modules/Module1
  {sync://srv2.ABCo.com:2647/Modules/Module1;1.1}
  {sync://srv2.ABCo.com:2647/Modules/Module1;1.2}
```

Example Showing Versions on the Trunk Branch (File-based)

```
This example shows the versions on the Trunk branch of top.v
(top.v;1.3 is in the local work area, so version information
for the Trunk branch is returned).
  dss> url versions top.v
  sync://holzt:2647/Projects/Sportster/top/top.v;1.1
  sync://holzt:2647/Projects/Sportster/top/top.v;1.2
  sync://holzt:2647/Projects/Sportster/top/top.v;1.3
```

Example Showing Versions on File Vault Object (File-based)

```
This example shows the versions of samp.asm on the "rel20"
branch:
  stcl> url versions [url vault samp.asm]rel20:Latest
 {sync://localhost/Projects/Sportster/code/samp.asm;1.2.1.1}
 {sync://localhost/Projects/Sportster/code/samp.asm;1.2.1.2}
```

url view

url view Command

NAME

```
url view            - Returns persistent view list for workspace module
```

DESCRIPTION

```
This command returns the persistent view list set on a workspace
module instance.
```

SYNOPSIS

```
url view  [--] <Workspace Module>
```

ARGUMENTS

- Workspace Module

Workspace Module

```
<workspace module>  Workspace module instance name or workspace
                    directory name containing the module.
```

OPTIONS

```
none
```

RETURN VALUE

```
For a workspace module instance DesignSync returns the persistent view
name or list.

If there is no persistent view for the workspace or the module does
not exist in the workspace,  DesignSync returns an empty string ("").
```

SEE ALSO

```
setview
```

EXAMPLES

- Example Showing A View

Example Showing A View

```
This example shows a technical writer's workspace, which has a Doc
view applied to the code module so the writer can populate only the
documentation.

dss> url view Chip%0
 Doc
```

TCL Interface

auto_mkindex

auto_mkindex Command

NAME

auto_mkindex - Indexes stcl procedures

DESCRIPTION

By placing your stcl procedures in designated site-wide or project-level tcl directories, DesignSync indexes them so that users at your site can autoload the procedures only when they need them. In some cases, you might want to generate the indexes manually using the auto_mkindex command. The indexes you create are not accessible to currently running DesignSync clients, but you can issue the auto_reset command in a DesignSync client to make the procedures in the indexes accessible. Thus, using auto_mkindex and auto_reset helps you efficiently debug your stcl procedures.

The auto_mkindex command creates the array of stcl procedures used by the current Tcl interpreter. These procedures are stored in the auto_index array and are loaded by the Tcl autoloader when users at your site invoke them. These autoloaded stcl procedures are only accessible in the stcl mode of DesignSync. If the auto_index generates successfully, a file named tclIndex is created in the site or custom tcl directory.

For your stcl procedures to be autoloaded, you must store the procedure files in the site-wide or project-level tcl directory. Thus, the <directory_name> argument must be the absolute path to the site or project tcl directory:

* For site-wide stcl files: <SYNC_SITE_CUSTOM>/share/client/tcl

* For project-level stcl files: <SYNC_PROJECT_CFGDIR>/tcl

SYNC_PROJECT_CFGDIR has no default; no project information is loaded if this environment variable is not set. SYNC_SITE_CUSTOM resolves to <SYNC_CUSTOM_DIR>/site.

Note: SYNC_SITE_CUSTOM is equivalent to <SYNC_CUSTOM_DIR>/site; if SYNC_SITE_CUSTOM is not set, but SYNC_CUSTOM_DIR is set, DesignSync will still access the site-wide stcl files.
You must have write access to the site or client tcl directory for the auto_index to be generated.

If you specify a file with the <file> argument, you must specify

```
the entire name of the file, including the .tcl extension.  If you
do not specify a file, auto_mkindex adds procedures in all files
with .tcl extensions in the specified directory to the auto_index
array.
```

SYNOPSIS

```
auto_mkindex <directory_name> [<file>...]

Note: This command is supported only in stcl mode.
```

RETURN VALUE

```
none
```

SEE ALSO

```
parray auto_index, auto_reset
```

EXAMPLES

```
This example adds the procedures in the site tcl directory to the
Tcl index.  In this example, the procedure, popasic, is included in
a file, revcmds.tcl in the site tcl directory,
<SYNC_DIR>/custom/site/share/client/tcl. The example also shows how
to use auto_reset to make the newly added procedure accessible in
the current session.

stcl>  auto_mkindex "c:\\Program Files\\Synchronicity\\DesignSync\\
       custom\\site\\share\\client\\tcl"
stcl> auto_reset
       0
stcl> popasic
       ...
```

auto_reset

auto_reset Command

NAME

```
auto_reset          - Resets the Tcl autoload index
```

DESCRIPTION

This command lets you use newly indexed stcl procedures without restarting your DesignSync client. Use the auto_mkindex command to index new stcl procedures. Then issue the auto_reset command to make the new procedures accessible in your current DesignSync session.

You must have write access to the site-wide and project-level tcl directories, as well as the tclIndex files in those directories, for the indexes to be reloaded. The site and project tcl directories are located as follows:

* For site-wide stcl files: <SYNC_SITE_CUSTOM>/share/client/tcl

* For project-level stcl files: <SYNC_PROJECT_CFGDIR>/tcl

SYNOPSIS

auto_reset

Note: This command is supported only in stcl mode.

RETURN VALUE

0 if the indexes are loaded successfully; 1 otherwise.

SEE ALSO

auto_mkindex, parray auto_index

EXAMPLES

This example shows how to use auto_reset to make a newly added procedure accessible in the current session.

```
stcl> auto_mkindex "c:\\Program Files\\Synchronicity\\DesignSync\\
      custom\\site\\share\\client\\tcl"
stcl> auto_reset
      0
stcl> popasic
      ...
```

gets

gets Command

NAME

```
gets                - Reads a string from a file
```

DESCRIPTION

```
This command is the Tcl gets command with Synchronicity
extensions. Refer to a Tcl language reference manual for
a full description of the standard gets command.

Synchronicity has extended the gets command to  a -prompt
option that you use to specify a prompt string.
```

SYNOPSIS

```
See a Tcl language reference manual.
```

EXAMPLES

```
The following example demonstrates the -prompt option, which
pops up a simple dialog box (when run from the graphical
interface) and prompts the user for input:

  set name [gets stdin -prompt "Enter your name:"]
```

parray auto_index

parray auto_index Command

NAME

```
parray auto_index   - Lists the autoloaded stcl procedures
```

DESCRIPTION

```
This command returns the array of stcl procedures in the current
Tcl interpreter.  These procedures are stored in the auto_index
```

```
array and are loaded by the Tcl autoloader when you invoke them.
You can add procedures to the Tcl interpreter by storing them in a
designated tcl directory and then invoking a DesignSync client or
by indexing them manually using the auto_mkindex command.

If you use auto_mkindex to manually index the new procedures, issue
the auto_reset command to make the auto_index accessible in the
current session.  The autoloaded stcl procedures are only
accessible in the stcl mode of DesignSync.
```

SYNOPSIS

```
parray auto_index

Note: This command is supported only in stcl mode.
```

RETURN VALUE

```
An array of stcl procedures and their source files.
```

SEE ALSO

```
auto_mkindex, auto_reset
```

EXAMPLES

```
This example lists the stcl procedures currently indexed as part
of the Tcl interpreter:

  stcl> parray auto_index

  # auto_index(::safe::interpAddToAccessPath)= source {c:/
              Program Files/Synchronicity/DesignSync/share/tcl/
              library/safe.tcl}
  # auto_index(::safe::interpConfigure) = source {c:/Program Files/
              Synchronicity/DesignSync/share/tcl/library/safe.tcl}
  ...
  ...
  ...
  # auto_index(parray)                  = source {c:/Program Files/
              Synchronicity/DesignSync/share/tcl/library/parray.tcl}
  # auto_index(pkg_mkIndex)             = source {c:/Program Files/
              Synchronicity/DesignSync/share/tcl/library/init.tcl}
  # auto_index(pop)                     = source {c:/Program Files/
              Synchronicity/DesignSync/custom/site/share/client/tcl/
              autoload.tcl}
```

```
...
...
```

puts

puts Command

NAME

```
puts                - Writes a string to a file
```

DESCRIPTION

```
This command is the Tcl 'puts' command. Refer to a Tcl
language reference manual for a full description of the
standard 'puts' command.

IMPORTANT:
  In previous software versions, the 'puts' command was
  extended to support two special file identifiers, 'log'
  and 'trace'.  These file identifiers are no longer
  supported.  The Synchronicity 'puts' command is now the
  standard Tcl 'puts' command.  You can use 'puts stderr'
  in client scripts to display output on the screen.  You
  can use 'puts stderr' in server scripts to channel the
  output to the server's error log,
  $SYNC_CUSTOM_DIR/servers/<host>/<port>/logs/error_log.
  Migrate existing scripts by removing instances of the
  'log' and 'trace' identifiers.
```

SYNOPSIS

```
See a Tcl language reference manual.
```

SEE ALSO

```
log
```

rstcl

rstcl Command

NAME

```
rstcl               - Runs server-side stcl scripts
```

DESCRIPTION

```
This command runs server-side stcl scripts from DesignSync
clients. You can also execute server-side scripts by passing a URL
to the SyncServer from your browser. See the 'server-side' topic or
the ProjectSync User's Guide for details.

You run client-side scripts using the DesignSync run command or the
Tcl source command. The choice of whether to implement a script as
client-side or server-side depends on what you are trying to
accomplish. You can use client scripts to automate user tasks or
implement enhancements to the built-in user command set. You create
server-side scripts for any of the following reasons:
  - To set server-wide policies (such as triggers or access controls)
  - To create server customizations (such as customized ProjectSync
    panels or data sheets)
  - To reduce the amount of client/server traffic that a
    client-side script accessing vault data would require
  - To execute commands that are only available as server-side
    commands (such as 'access reset' and most ProjectSync commands)

When you execute a script with rstcl, the SyncServer looks for the
specified script in the following locations (in the order listed):

  1. Server-specific Tcl scripts (UNIX only):
     <SYNC_CUSTOM_DIR>/servers/<host>/<port>/share/tcl

  2. Site-wide Tcl scripts:
     <SYNC_CUSTOM_DIR>/site/share/tcl

  3. Enterprise-wide Tcl scripts:
     <SYNC_CUSTOM_DIR>/enterprise/share/tcl

  4. Synchronicity-provided Tcl scripts:
     <SYNC_DIR>/share/tcl

     Do not put scripts in <SYNC_DIR>/share/tcl because this
     directory is reserved for Synchronicity scripts, and your
     scripts may be overwritten when you upgrade your
     Synchronicity software.

rstcl requests mutually exclude each other. I.e. They all acquire the
same exclusive lock, named smdSrvrMetaDataLock. If you analyze your
script and know it to be safe to run in parallel with other scripts,
you may release the exclusive lock from within your script by using
'url syslock -release smdSrvrMetaDataLock'. If your script reads or
writes an external file, it is probably not parallelizable. rstcl
requests and panel= requests (invoked via ProjectSync) never mutually
exclude each other; panel requests are entirely independent of rstcl's
lock.
```

```
Notes:
- If you make modifications to your script, use the ProjectSync
  Reset Server menu option to force the SyncServer to reread your
  script.
- When specifying 'sync:' URLs within scripts that are run on the
  server, do not specify the host and port. For example, specify:
    sync:///Projects/Asic
  not
    sync://holzt:2647/Projects/Asic
  Because the script is run on the server itself, host:port
  information is unnecessary and is stripped out by the
  server, which may lead to incorrect behavior during object-name
  comparisons.
- The SYNC_ClientInfo variable is not defined when running
  server-side scripts with rstcl -- you must use the browser-based
  invocation. All other SYNC_* variables (SYNC_Host, SYNC_Port,
  SYNC_Domain, SYNC_User, and SYNC_Parm if parameters are passed
  into the script) are available when using rstcl.
```

SYNOPSIS

```
rstcl [-output <file>] -server <serverURL> -script <script>
      [-urlparams <name>=<value>[&<name>=value[...]]]
```

OPTIONS

-output

```
 -output <file>     Specifies the file to which script output is
                    written. If omitted, the output is displayed.
```

-server

```
-server <serverURL> Specifies the URL of the SyncServer that will
                    execute the script. Specify the URL as follows:
                      sync://<host>[:<port>]
                    where 'sync://' is required, <host> is
                    the machine on which the SyncServer is
                    installed, and <port> is the SyncServer port
                    number (if omitted, defaults to 2647/2679). For
                    example, you might specify the following:
                      -server sync://serv1.abco.com:1024
```

-script

```
-script <script>    Specifies the name of the script to be
                    executed. This script must be in one of the Tcl
                    script directories on the SyncServer specified
                    by the -server option. The Tcl directories are
                    (in the order in which they are searched):

                    1. Server-specific Tcl scripts (UNIX only):
                    <SYNC_DIR>/custom/servers/<host>/<port>/share/tcl

                    2. Site-wide Tcl scripts:
                    <SYNC_DIR>/custom/site/share/tcl

                    3. Enterprise-wide Tcl scripts:
                    <SYNC_DIR>/custom/enterprise/share/tcl

                    4. Synchronicity-provided Tcl scripts:
                    <SYNC_DIR>/share/tcl

                    The script can contain Tcl constructs and
                    Synchronicity commands, including server-side only
                    commands.
```

-urlparams

```
-urlparams <params> Specifies the parameters that are passed into
                    the script. Specify each parameter as a
                    name/value pair separated by an equal sign (=),
                    and separate multiple parameters with an
                    ampersand (&):
                      <param1>=<value1>&<param2>=<value2>...
                    For example:
                      -urlparams Name=Joe&IDNum=1234

                    Parameters are passed into the script using the
                    global variable SYNC_Parm, which is a Tcl
                    array. The array keys are the names of the
                    parameters. To access the value of a parameter
                    from within the script, use the following syntax:

                      $SYNC_Parm(<param_name>)

                    For example, the following Tcl line would
                    display the value of the 'name' parameter:

                      puts "The name is: $SYNC_Parm(name)"

                    Note: If any parameter name or value
                    contains whitespace, surround the entire
                    parameter list with double quotes. For example:
```

```
                    -urlparams "name=Joe Black&IDNum=1234"
```

RETURN VALUE

```
o If -output is not specified, returns (and displays) the script
  output.
o If -output is specified, output is written to the specified file
  and the return value is an empty string.

If the script has an error, a Tcl exception is thrown from the
client side and the Tcl stack trace is output. Proper usage
for handling exceptions would be to provide an exception handler
when you use the rstcl command:
  if [catch {rstcl -server ...} result] {
      # Something bad happened.
      # 'result' contains the output generated by the script
      #   prior to the error and the Tcl stack trace.
  } else {      # All is fine.
      # 'result' contains whatever output is generated
      # by the script.
  }

If the -output option to the rstcl command was specified, then
the exception is still thrown, but the script output and Tcl stack
trace are written to the specified output file.
```

SEE ALSO

```
server-side, run, url syslock
```

EXAMPLES

```
A common use of rstcl is to run the 'access reset' command, which
restarts the SyncServer. See the 'access reset' command for details.

Most ProjectSync-related scripts must be run on the server and
could therefore use rstcl. This example creates a ProjectSync note
using the 'note create' command, which is a server-side only
command, and displays the URL of the new note. This output is then
returned to the rstcl command in callNoteCreate.tcl.

 1.In the <SYNC_CUSTOM_DIR>/site/share/tcl directory on the
 holzt:2647 server is the noteCreate.tcl script, which contains
 the following:

   set noteUrl [note create -type Note \
     [list Title $SYNC_Parm(title)] [list Body $SYNC_Parm(body)] \
     [list Author $SYNC_Parm(author)] ]
   puts "$noteUrl"
```

```
2. On the client side, the callNoteCreate.tcl script provides an
exception catcher in case the noteCreate.tcl script fails.

  if [catch {rstcl -server sync://holzt:2647 -script noteCreate.tcl \
       -urlparams "author=Goss&title=This is a note.&body=New note."} \
       result] {
      puts "Couldn't create the note!"
  } else {
      puts "Created note: $result"
  }

3. From stcl, run the client script:

     stcl> source callNoteCreate.tcl
     Created note: sync:///Note/SyncNotes/Note/3

   You could also run the rstcl command directly from the command
   line (no exception catcher). Doing so creates a second note:

     stcl> rstcl -server sync://holzt:2647 -script noteCreate.tcl \
           -urlparams "author=Goss&title=Another note.&body=New note."
     sync:///Note/SyncNotes/Note/4
```

run

run Command

NAME

```
run                 - Executes a DesignSync command file or stcl script
```

DESCRIPTION

```
This command will execute the DesignSync or Tcl commands contained
in the specified file. Specify the file with a relative or absolute
path, not as a URL. If the extension of the file is ".tcl", the
script is run in stcl mode, irrespective of the current mode (dss
or stcl). Otherwise, the script is run in dss/dssc mode
irrespective of your current mode. From stcl/stclc, using the run
command is the same as using the Tcl source command except that the
run command does not return the script's return value.

If you do not specify a path to the command file, the run command
looks in the default log directory. By default, the default log
directory is your home directory (as defined by $HOME on UNIX or
your user profile, which is managed by the User Manager tool, on
Windows platforms).  You can change the default using the
-defaultdir option to either the log or the run command.

The batch file may be created in a text editor or captured with
```

```
the log command. The '#' character in column 1 treats the
remainder of the line as a comment.

Note: Use the rstcl command to execute server-side stcl scripts.
```

SYNOPSIS

```
run [-defaultdir <dir>] [-dryrun] [-ignoreerrs] [-verbose]
    [--] [<filename>]
```

OPTIONS

- -defaultdir
- -dryrun
- -ignoreerrs
- -verbose
- --

-defaultdir

```
-defaultdir <dir>  Set the location of the default log directory,
                   which is also where DesignSync looks for
                   scripts. This value is saved between sessions.

                   Note: Specifying the defaultdir is mutually
                   exclusive with specifying a filename to run.
```

-dryrun

```
-dryrun            Like verbose, but do not execute. This option
                   is useful to verify the behavior of a command file
                   before executing it.
```

-ignoreerrs

```
-ignoreerrs        Continue executing the DesignSync command file even
                   if an error is encountered.

                   This option is not allowed when running stcl
                   (*.tcl) scripts. You must provide your own Tcl
                   exception handler to catch errors in your stcl
                   script. Use the Tcl 'catch' command.
```

-verbose

```
 -verbose           Print commands as they are executed.
```

--

```
 --                 Indicates that the command should stop looking for
                    command options. Use this option when arguments
                    to the command begin with a hyphen (-).
```

RETURN VALUE

```
 none
```

SEE ALSO

```
 log, rstcl
```

EXAMPLES

```
This example sets the default log directory, which is also where
DesignSync looks for scripts, to /home/goss/Projects, then executes
myscript.dss.

 dss> run -defaultdir /home/goss/Projects
 dss> run myscript.dss
```

Third-Party Integrations

DSDFII

addcdslib

addcdslib Command

NAME

```
addcdslib           - Specifies a cds.lib file
```

DESCRIPTION

```
This command adds a path to the search path that DesignSync
uses to locate Cadence cds.lib files. The cds.lib files map Cadence
libraries to their locations. For example, a cds.lib file might
contain the following:
  DEFINE TTL1  /home/TTLLibraries/TTL1
  DEFINE basic /usr1/CoreLibraries/basic

DesignSync uses these mappings to resolve dependencies that a
design object, in this case a Cadence cell view, has on other design
objects. Using the 'url relations' command, you could, for example,
write an stcl script that checks out a cell view and all of its
dependencies.

Specify the path argument to the addcdslib command as the absolute
path to the directory containing the cds.lib file (do not include
"cds.lib" as part of the specification).
```

SYNOPSIS

```
addcdslib <path>
```

OPTIONS

- --

--

```
--            Indicates that the command should stop looking for
              command options. Use this option when the path
              you specify begins with a hyphen (-).
```

SEE ALSO

```
url relations
```

EXAMPLES

```
This command shows how to resolve dependencies by using the
addcdslib command. The cds.lib file in /home/Libraries contains
the library definition for "basic", but not for "sample".

 stcl> url relations cmos_sch.sync.cds dependencies
   {<unrecognized alias>} basic:vdd/symbol.sync.cds
   {<unrecognized alias>} basic:gnd/symbol.sync.cds
   {<unrecognized alias>} sample:nmos/symbol.sync.cds
 stcl> addcdslib /home/Libraries
 stcl> url relations cmos_sch.sync.cds dependencies
   file:///home/tgoss/Projects/Cadence/basic/opin/symbol.sync.cds\
   basic:opin/symbol.sync.cds
   file:///home/tgoss/Projects/Cadence/basic/gnd/symbol.sync.cds\
   basic:gnd/symbol.sync.cds
   {<unrecognized alias>} sample:nmos/symbol.sync.cds
```

Administration

Access Control

ACAdmin Commands

acadmin

acadmin Command

NAME

```
acadmin             - Access Administrator Commands
```

DESCRIPTION

```
The Access Administrator tool (ACAdmin) provides a graphical web
interface to create, remove, maintain, and manage access controls.
Using the ACAdmin interface provides a simpler, more intuitive
interface to customizing the access controls for DesignSync. In
addition to the graphical web interface, DesignSync provides a set of
acadmin commands, prefixed by "acadmin" to use for scripting or other
acadmin maintenance. For more information on the functionality,
organization and usage of ACAdmin, see the ENOVIA Synchronicity
Access Control Guide.

In order to use any of the acadmin commands, the Access Control
Administrator must be enabled on the server. When any of these
commands are used to modify the ACAdmin configuration, the changes
made are not applied to the server until the acadmin reset command is
run.
```

SYNOPSIS

```
acadmin <acadmin_command> [acadmin_command_options>]

Usage: acadmin
[addgroup|addgroupusers|addobj|addusers|listcats|listcmds|listgroups|
listobjs|listperms|listusers|reset|rmgroup|rmgroupusers|rmobj|
rmusers|setcatperm]
```

EXAMPLES

```
See specific acadmin commands.
```

acadmin addgroup

acadmin addgroup Command

NAME

```
acadmin addgroup    - Create a new user group.
```

DESCRIPTION

```
This command creates user groups. These user groups are then
available to be assigned to command categories. Grouping users into
groups allows you to define and assign roles and maintain the roles
and functions easily even when individual group members change.

Notes:
 o Usually only the DesignSync administrators should have permission
   to create or modify user group definitions.

 o There are three dynamic (or virtual) user groups that should never
   be manipulated manually. They are All-Module-Owners,
   All-Project-Owners, and All-Server-Users. These groups are
   automatically generated when ACAdmin is reset. If you have made
   changes that affect these groups, you should perform an ACAdmin
   Reset.

This command is subject to ACAdmin access controls as defined in
AccessControl.aca. See the ENOVIA Synchronicity Access Control Guide
for details.

To use this command, the Access Control Administrator must be enabled
on the server being updated.

To apply the changes created by this command to the server, you must
reset the server using the acadmin reset command or ACAdmin Reset
from the DesignSync Web UI.
```

SYNOPSIS

```
acadmin addgroup -group <groupName> [-server <SyncURL>]
```

ARGUMENTS

- Sync URL

Sync URL

```
-server <SyncURL>   Specifies the option Sync URL as follows:
                    sync://<host>[:<port>] or
                    syncs://<host>[:<port>]
                    where 'sync://' or 'syncs://' are required,
                    <host> is the machine on which the SyncServer is
                    installed, and <port> is the SyncServer port
                    number (defaults to 2647/2679).  For example:
                    sync://apollo.spaceco.com:1024

                    If this argument is not specified, DesignSync
                    users the environment variable SYNC_SERVER_URL.

                    If there is no SYNC_SERVER_URL variable defined,
                    DesignSync uses the vault association set on the
                    current working directory. If there is no vault
                    or there are multiple vaults associated with the
                    directory, for example, when you have overlapping
                    modules, the command fails.
```

OPTIONS

- -group

-group

```
-group <groupName>  Name of the user group. This name should
                    correspond to DesignSync naming conventions. For
                    a list of reserved characters that should not be
                    used in the user group name, see the ENOVIA
                    Synchronicity DesignSync Data Manager User's
                    Guide: URL Syntax.

                    Tip: To allow you to easily differentiate between
                    user groups and individual users, you should
                    implement a naming convention such as prefixing
                    the user group with a standard prefix like Group-
                    or by creating group names in all capital
                    letters. For example:
                    DEVELOPERS
                    or
                    GROUP-Developers
```

RETURN VALUE

```
This command doesn't return any TCL values. If the command succeeds,
you'll receive a success message.  If the command fails, you'll
receive an error message explaining the failure.
```

SEE ALSO

```
acadmin addgroupusers, acadmin listgroups, acadmin rmgroup
```

EXAMPLES

- Example of Creating a Group in ACAdmin

Example of Creating a Group in ACAdmin

```
This example creates a documentation group in ACAdmin.

dss> acadmin addgroup -group GROUP-Doc -server \
sync://serv1.ABCo.com:2647

 Created 1 User Group(s)
```

acadmin addgroupusers

acadmin addgroupusers Command

NAME

```
acadmin addgroupusers - Add users to group
```

DESCRIPTION

```
This command adds defined users to an existing group. Both usernames
and group names are case sensitive. You can only add users to one
group at a time.

The group  being added to must already exist.  To create new groups,
use the acadmin addgroup command.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

To use this command, the Access Control Administrator must be enabled
on the server being updated.

To apply the changes created by this command to the server, you must
reset the server using the acadmin reset command or ACAdmin Reset
from the DesignSync Web UI.
```

SYNOPSIS

```
acadmin addgroupusers -group <groupName> -users <userlist>
                      [-server <SyncURL>]
```

ARGUMENTS

- Sync URL

Sync URL

```
-server <SyncURL>   Specifies the option Sync URL as follows:
                    sync://<host>[:<port>] or
                    syncs://<host>[:<port>]
                    where 'sync://' or 'syncs://' are required,
                    <host> is the machine on which the SyncServer is
                    installed, and <port> is the SyncServer port
                    number (defaults to 2647/2679).  For example:
                    sync://apollo.spaceco.com:1024

                    If this argument is not specified, DesignSync
                    users the environment variable SYNC_SERVER_URL.

                    If there is no SYNC_SERVER_URL variable defined,
                    DesignSync uses the vault association set on the
                    current working directory. If there is no vault
                    or there are multiple vaults associated with the
                    directory, for example, when you have overlapping
                    modules, the command fails.
```

OPTIONS

- -group
- -user

-group

```
-group <groupName>  Name of the group to which the users are being
                    added. This group must already exist.  Group
                    names are case sensitive.

                    If you are not sure what groups exist, use the
                    acadmin listgroup command to get a list of
                    existing groups.

                    Note: If the group is associated with an object,
                    you need to specify the fully extended group
                    name.  Fully extended group name is specified as:

                    <GroupName>@sync:///<ServerObjectPath>
                   Where <GroupName> is the case sensitive name of
                    the group.
                   <ServerObjectPath> is the path to the object on
                    which the group was created for example:
                   Projects/[<ProjectFolder>/...]ProjectName
                   Modules/[<Category>/...]ModuleName
```

-user

```
-user <userlist>    A comma separated list of users to associate with
                    the group.  The users must already exist.  You
                   may also use the special user definitions
                   provided by DesignSync.  For more information on
                   the special users, see the Access Administration
                   Guide.
```

RETURN VALUE

```
This command does not return any TCL values.  When successful, this
command returns a success message.  If the command fails, it displays
an error message explaining the failure.
```

SEE ALSO

```
acadmin addgroup, acadmin addusers, acadmin listusers,
acadmin listgroups, acadmin rmgroup, acadmin rmgroupusers
```

EXAMPLES

Example of a User defined for a Usergroup

```
This example shows adding a user to a usergroup.

dss> stcl> acadmin addgroupusers -group DocWriters -users rsmith \
     -server sync://serv1.ABCo.com:30126

Updated 1 User Group(s)
```

Example of Adding a User to a Usergroup defined for an Object

```
This example shows adding a user to a usergroup when the usergroup is
defined for a specific object, in this example, it is the module
XLP-12Pro in the ChipDesign category.

dss> acadmin addgroupusers -group \
  chipDevelopers@sync:///Modules/ChipDesign/XLP-12Pro  -users \
  thopkins -server sync://serv1.ABCo.com:30126
```

```
Updated 1 User Group(s)
```

acadmin addobj

acadmin addobj Command

NAME

```
acadmin addobj      - Manage the given object with ACAdmin
```

DESCRIPTION

```
This command adds permissions for all the existing categories for
specified object to the acadmin configuration files. Using the
setcatperm command, you can then modify the permissions for each
category as needed, granting finer control for the specified object.

The object does not need to exist in order to be added. For example,
if you have a list of modules be created, you can define the access
permissions first and then create the modules as needed.

Note: When this command is run, it assigns the permission(s) selected
to all categories that exist.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

To use this command, the Access Control Administrator must be enabled
on the server being updated.

To apply the changes created by this command to the server, you must
reset the server using the acadmin reset command or ACAdmin Reset
from the DesignSync Web UI.
```

SYNOPSIS

```
acadmin addobj -object <objectURL> [-permission <permission>]
               [-server <SyncURL>]
```

ARGUMENTS

- Sync URL

Sync URL

```
-server <SyncURL>   Specifies the option Sync URL as follows:
```

```
sync://<host>[:<port>] or
syncs://<host>[:<port>]
where 'sync://' or 'syncs://' are required,
<host> is the machine on which the SyncServer is
installed, and <port> is the SyncServer port
number (defaults to 2647/2679).  For example:
sync://apollo.spaceco.com:1024

If this argument is not specified, DesignSync
users the environment variable SYNC_SERVER_URL.

If there is no SYNC_SERVER_URL variable defined,
DesignSync uses the vault association set on the
current working directory. If there is no vault
or there are multiple vaults associated with the
directory, for example, when you have overlapping
modules, the command fails.
```

OPTIONS

-object

```
-object <objectURL> Enter the Sync URL of the object on which to set
                    the permissions. You can set the permissions for
                    any object on the server. You may specify objects
                    as follows:

                    sync:///
                    Controls the default permissions for all
                   object-independent actions on the server.  For
                   example, user creation and removal actions do not
                   depend on specific server objects. (Default)

                    sync:///*
                    Controls the default permissions for all
                    object-dependent actions on the server. For
                    example, browsing objects on the server, creating
                    or modifying modules, etc. depend on having
                    access to related server objects.

                    sync:///Projects[/ProjectName]/[pathtoObject]
                    Controls the permission for the specified project
                    or the entire project area on the server.

                    sync:///Modules[/Category [...]][/ModName]
                    Controls the permission for the specified module,
                    category, or entire module area on the server.

                    Notes:
                    You can use wildcards, such as * to specify all
```

matching objects within the specified path, for example: sync:///Modules/Chip/*.c controls access to all .c files within the module Chip. If Chip was a category and you wanted to specify all modules within the category, you could specify the URL either of the following ways:
sync:///Modules/Chip/*
sync:///Modules/Chip

The sync:/// and sync:///* URLs are generic Sync URLs that can be used to provide default access for the server.

-permission

-permission <permission>	Enter a comma separated list of defined permissions to associate with the object. ALL, LIST, EXCLLIST, NONE, SERVDEF When LIST or EXCLLIST are used, you can specify a user or userlist for whom to set the permissions.

RETURN VALUE

This command doesn't return any TCL values. If the command succeeds, you'll receive a success message for each category associated with the object. If the command fails, you'll receive an error message explaining the failure.

SEE ALSO

acadmin addgroup, acadmin addgroupusers, acadmin rmobj

EXAMPLES

- Example of Adding the Permissions Categories to A Server Object

Example of Adding the Permissions Categories to A Server Object

This example shows how to create categories for a server object. The command creates all the categories that exist on the server in the acadmin configuration file.

```
dss> acadmin addobj -object sync:///Modules/Chip/ALU -permission ALL \
     -server sync://serv1.ABCo.com:30126
```

```
Category ACAProjectDefs created
Category ADMIN-MODULE created
Category BROWSE created
Category DS-PROJADMIN created
Category DS-READ created
Category DS-WRITE created
Object sync:///Modules/Chip/ALU has been added
```

acadmin addusers

acadmin addusers Command

NAME

```
acadmin addusers     - Add a user(s) to specified Object and Category
```

DESCRIPTION

```
This command adds one or more users to a specified category of
permissions assigned to an object. This allows you to modify the list
of users assigned to a permissions category as needed.

 Note: The specified user does not need to exist when this
 command is run; it can be created later. For more information on
 creating users in DesignSync see the DesignSync Administrator's
 Guide.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

To use this command, the Access Control Administrator must be enabled
on the server being updated.

To apply the changes created by this command to the server, you must
reset the server using the acadmin reset command or ACAdmin Reset
from the DesignSync Web UI.
```

SYNOPSIS

```
acadmin addusers -category <categoryName> -object <objectURL>
                 -users <userlist> [-server <SyncURL>]
```

ARGUMENTS

- Sync URL

Sync URL

```
-server <SyncURL>   Specifies the option Sync URL as follows:
                    sync://<host>[:<port>] or
                    syncs://<host>[:<port>]
                    where 'sync://' or 'syncs://' are required,
                    <host> is the machine on which the SyncServer is
                    installed, and <port> is the SyncServer port
                    number (defaults to 2647/2679).  For example:
                    sync://apollo.spaceco.com:1024

                    If this argument is not specified, DesignSync
                    users the environment variable SYNC_SERVER_URL.

                    If there is no SYNC_SERVER_URL variable defined,
                    DesignSync uses the vault association set on the
                    current working directory. If there is no vault
                    or there are multiple vaults associated with the
                    directory, for example, when you have overlapping
                    modules, the command fails.
```

OPTIONS

-category

```
-category           Name of the command category associated with the
<categoryName>      user. The command category must exist already.
```

-option

```
-object <objectURL> Sync URL of the object associated with the user.
                    The object or object special URL (for example,
                    sync:///) must already exist.
```

-user

```
-user <userlist>    A comma separated list of users to associate with
                    the category and object.
```

RETURN VALUE

```
This command does not return a TCL value. The commands displays a
```

```
success message when successful or an appropriate error if the
command fails.
```

SEE ALSO

```
acadmin addgroup, acadmin addgroupusers, acadmin addobj,
acadmin rmusers
```

EXAMPLES

- example_adduser_multiple

```
This example shows adding three users to a category for an object.

dss> acadmin addusers -category ADMIN-MODULE -object \
sync:///Modules/Chip/ALU -users rsmith,thopkins,chipAdmin  \
-server sync://lwvrh17mon:30126

Category ADMIN-MODULE updated
```

acadmin listcats

acadmin listcats Command

NAME

```
listcats            - Lists all categories defined on a server
```

DESCRIPTION

```
The listcats command lists all the categories defined on the
specified server.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

To use this command, the Access Control Administrator must be enabled
on the server being updated.
```

SYNOPSIS

```
acadmin listcats [-server <SyncURL>]
```

ARGUMENTS

- Sync URL

Sync URL

```
-server <SyncURL>   Specifies the option Sync URL as follows:
                    sync://<host>[:<port>] or
                    syncs://<host>[:<port>]
                    where 'sync://' or 'syncs://' are required,
                    <host> is the machine on which the SyncServer is
                    installed, and <port> is the SyncServer port
                    number (defaults to 2647/2679).  For example:
                    sync://apollo.spaceco.com:1024

                    If this argument is not specified, DesignSync
                    users the environment variable SYNC_SERVER_URL.

                    If there is no SYNC_SERVER_URL variable defined,
                    DesignSync uses the vault association set on the
                    current working directory. If there is no vault
                    or there are multiple vaults associated with the
                    directory, for example, when you have overlapping
                    modules, the command fails.
```

RETURN VALUE

```
This command does not return a TCL value. When successful, this
command lists the categories available on the server. If the command
fails, it displays an error message explaining the failure.
```

SEE ALSO

```
acadmin addusers, acadmin listcmds, acadmin listperms,
acadmin setcatperm
```

EXAMPLES

- Example of Listing the Categories for a Server

Example of Listing the Categories for a Server

```
This example shows listing the categories for a server. This server
has no custom categories, so the listing shows only the default
categories that come with ACAdmin.
dss> acadmin listcats -server sync://serv1.ABCo.com:2647

 ACAProjectDefs
```

```
ADMIN-MODULE
BROWSE
DS-PROJADMIN
DS-READ
DS-WRITE
Mirrors
PS-Read
PS-Write
SRV-ADMIN
```

acadmin listcmds

acadmin listcmds Command

NAME

```
acadmin listcmds    - Lists all commands by category
```

DESCRIPTION

```
The listcmds command lists all the commands associated with the
defined categories. If you are interested in the commands associated
with a specific category, you can use the -category option to limit
the results to a single category.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

To use this command, the Access Control Administrator must be enabled
on the server being updated.
```

SYNOPSIS

```
acadmin listcmds [-category <categoryName>] [-server <SyncURL>]
```

ARGUMENTS

- Sync URL

Sync URL

```
-server <SyncURL>   Specifies the option Sync URL as follows:
                    sync://<host>[:<port>] or
                    syncs://<host>[:<port>]
                    where 'sync://' or 'syncs://' are required,
                    <host> is the machine on which the SyncServer is
                    installed, and <port> is the SyncServer port
```

```
                    number (defaults to 2647/2679).  For example:
                    sync://apollo.spaceco.com:1024

                    If this argument is not specified, DesignSync
                    users the environment variable SYNC_SERVER_URL.

                    If there is no SYNC_SERVER_URL variable defined,
                    DesignSync uses the vault association set on the
                    current working directory. If there is no vault
                    or there are multiple vaults associated with the
                    directory, for example, when you have overlapping
                    modules, the command fails.
```

OPTIONS

- -category

-category

```
-category           Name of the command category associated with the
<categoryName>      user. The command category must exist already.

                    Note: You can only specify a single category.
```

RETURN VALUE

```
Does not return a TCL value.  Displays an alphabetized list of
categories and the commands within each category.
```

SEE ALSO

```
acadmin addusers, acadmin listcats, acadmin listperms,
acadmin setcatperm
```

EXAMPLES

- Example Showing Listing Commands for All Categories on the Server

Example Showing Listing Commands for All Categories on the Server

```
This example shows the list of command associated with the categories
for a server. This server has no custom categories, so the listing
shows only the default categories that come with ACAdmin.

dss> acadmin listcmds -server sync://serv1.ABCo.com:2647

 ACAProjectDefs {AcaProjCatPrmDef AcaProjUserGroupDef}
```

```
ADMIN-MODULE {ChangeCommentAll DeleteFolder DeleteVault
DeleteVersion ExportModule FreezeModule ImportModule MemberUnlockAll
Mkmod Move MoveModule Rmalias Rmconf Rmedge Rmmod Rollback SetOwner
SwitchLocker TagRelease UnfreezeModule UnlockAll}
BROWSE BrowseServerObj
DS-PROJADMIN {ChangeCommentAll DeleteFolder DeleteVault
DeleteVersion MakeBranch Move SetOwner SwitchLocker TagRelease
UnlockAll}
DS-READ CheckoutNoLock
DS-WRITE {Addhref ChangeComment Checkin CheckoutLock HcmUpgrade Lock
MakeBranch MakeBranchTrunk MakeFolder MemberUnlock Mkedge Retire
Rmhref Tag Unlock Unretire}
Mirrors Mirrors
PS-Read {AcaViewDef BrowseServer EditUser EmailSubscribe ViewNote}
PS-Write {AddNote AddPSReport CreateConfig DeleteNote DeletePSReport
EditNote EditNoteAttachments EditUser ModifyConfig
ModifyNoteProperty ReviseNoteHistory SetNoteProperty UnlockNote
ViewNote}
SRV-ADMIN {ACAActions AddDevelopmentInstance Addlogin AddMirror
AddProject AddTrigger AddUser AdministrateNoteTypes
AdministrateServer DeleteConfig DeleteDevelopmentInstance
DeleteMirror DeleteProject DeletePSReportAll DeleteTrigger
DeleteUser EditAllUser EditMirror EditTrigger EmailAllSubscribe
EmailMgrAdmin ExportProject ImportProject ModifyDevelopmentInstance
ModifyMirror ModifyProject PrimaryMirrorFetch ResetAccessControls
Rmlogin Showlogins Suspend TransferFile ViewMirror}
```

acadmin listgroups

acadmin listgroups Command

NAME

```
acadmin listgroups  - lists the defined groups and their users
```

DESCRIPTION

```
The acadmin listgroups command lists all defined groups. All groups
are available server-wide. Groups can be specified as associated with
an object, however this association is non-binding and purely
intended as a guide to the user to indicate how the group should be
used.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

To use this command, the Access Control Administrator must be enabled
on the server being updated.
```

SYNOPSIS

```
acadmin listgroups [-group <groupName>] [-server <SyncURL>]
```

ARGUMENTS

- Sync URL

Sync URL

```
-server <SyncURL>   Specifies the option Sync URL as follows:
                    sync://<host>[:<port>] or
                    syncs://<host>[:<port>]
                    where 'sync://' or 'syncs://' are required,
                    <host> is the machine on which the SyncServer is
                    installed, and <port> is the SyncServer port
                    number (defaults to 2647/2679).  For example:
                    sync://apollo.spaceco.com:1024

                    If this argument is not specified, DesignSync
                    users the environment variable SYNC_SERVER_URL.

                    If there is no SYNC_SERVER_URL variable defined,
                    DesignSync uses the vault association set on the
                    current working directory. If there is no vault
                    or there are multiple vaults associated with the
                    directory, for example, when you have overlapping
                    modules, the command fails.
```

OPTIONS

- -group

-group

```
-group <groupName>  Name of the group you want to view the users
                    associated with.

                   Note: If the group was associated with an object,
                    you need to specify the fully extended group
                    name.  Fully extended group name is specified as:

                    <GroupName>@sync:///<ServerObjectPath>
                   Where <GroupName> is the case sensitive name of
                    the group.
                   <ServerObjectPath> is the path to the object on
                    which the group was created for example:
                   Projects/[<ProjectFolder>/...]ProjectName
                   Modules/[<Category>/...]ModuleName
```

RETURN VALUE

```
This command does not return any TCL values. If the command is
successful it will display a list of groups, alphabetically, followed
by a list of users.
```

SEE ALSO

```
acadmin addgroup, acadmin addgroupusers, acadmin rmgroup,
acadmin rmgroupusers
```

EXAMPLES

Example Showing a List of All Groups

```
This example shows the list of all groups on the server.  Groups
associated with a specific object are shown with the object path.

dss> acadmin listgroups -server sync://serv1.ABCo.com:30126

 DocWriters mhopkins
 chipDevelopers@sync:///Modules/ChipDev/XLP-12Pro {rsmith thopkins}
```

Example Showing the Users for a Specified Group

```
This example shows the list for the chipDevelopers group on the
Module object XLP-12Pro.  Note  that when you specify a group that is
on an object, you must specify the full group path as
<group>@<relativeObjectPath>, as shown.

dss> acadmin listgroups -group \
chipDevelopers@sync:///Modules/ChipDev/XLP-12Pro -server \
sync://serv1.ABCo.com:30126

 chipDevelopers@sync:///Modules/ChipDev/XLP-12Pro {rsmith thopkins}
```

acadmin listobjs

acadmin listobjs Command

NAME

```
acadmin listobjs    - lists all acadmin managed objects on the server
```

DESCRIPTION

```
This command lists all the objects that are managed in the acadmin
configuration files. The objects can be added with the acadmin
addobjs command and modified with the setcatperm command.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

To use this command, the Access Control Administrator must be enabled
on the server being updated.
```

SYNOPSIS

```
acadmin listobjs [-server <SyncURL>]
```

ARGUMENTS

- Sync URL

Sync URL

```
-server <SyncURL>   Specifies the option Sync URL as follows:
                    sync://<host>[:<port>] or
                    syncs://<host>[:<port>]
                    where 'sync://' or 'syncs://' are required,
                    <host> is the machine on which the SyncServer is
                    installed, and <port> is the SyncServer port
                    number (defaults to 2647/2679).  For example:
                    sync://apollo.spaceco.com:1024

                    If this argument is not specified, DesignSync
                    users the environment variable SYNC_SERVER_URL.

                    If there is no SYNC_SERVER_URL variable defined,
                    DesignSync uses the vault association set on the
                    current working directory. If there is no vault
                    or there are multiple vaults associated with the
                    directory, for example, when you have overlapping
                    modules, the command fails.
```

RETURN VALUE

```
This command does not return any TCL values.  When successful, this
```

```
command returns a list, in alphabetical order, of all the objects
managed by acadmin.  If the command fails, it returns an error
message explaining the failure.
```

SEE ALSO

```
acadmin addobj, acadmin rmobj
```

EXAMPLES

- Example of Listing Objects Managed by ACAdmin

Example of Listing Objects Managed by ACAdmin

```
This example shows a list of all the objects managed by ACAdmin.

dss> acadmin listobjs -server sync://serv1.ABCo.com:30127

 sync:///
 sync:///Modules/ChipDev/ALU
 sync:///Modules/ChipDev/XLP-12Pro
```

acadmin listperms

acadmin listperms Command

NAME

```
acadmin listperms   - list all permissions for the object or category
```

DESCRIPTION

```
This command shows the permissions for all the objects that are
managed with ACAdmin.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

To use this command, the Access Control Administrator must be enabled
on the server being updated.
```

SYNOPSIS

```
acadmin listperms [-category <categoryName>] -object <ObjectURL>
[-server <SyncURL>]
```

ARGUMENTS

- Sync URL

Sync URL

```
-server <SyncURL>   Specifies the option Sync URL as follows:
                    sync://<host>[:<port>] or
                    syncs://<host>[:<port>]
                    where 'sync://' or 'syncs://' are required,
                    <host> is the machine on which the SyncServer is
                    installed, and <port> is the SyncServer port
                    number (defaults to 2647/2679).  For example:
                    sync://apollo.spaceco.com:1024

                    If this argument is not specified, DesignSync
                    users the environment variable SYNC_SERVER_URL.

                    If there is no SYNC_SERVER_URL variable defined,
                    DesignSync uses the vault association set on the
                    current working directory. If there is no vault
                    or there are multiple vaults associated with the
                    directory, for example, when you have overlapping
                    modules, the command fails.
```

OPTIONS

- -category
- -object

-category

```
-category           Name of the command category associated with the
<categoryName>      user. The command category must exist already.

                    Note: You can only specify a single category.
```

-object

```
-object <objectURL> Enter the Sync URL of the object for which you
                     are viewing the permissions. You can view the
                    permissions for any object on the server. You may
                    specify objects as follows:

                     sync:///
                     Controls the default permissions for all
                    object-independent actions on the server.  For
                    example, user creation and removal actions do not
                    depend on specific server objects. (Default)
```

```
sync:///*
Controls the default permissions for all
object-dependent actions on the server. For
example, browsing objects on the server, creating
or modifying modules, etc.  depend on having
access to related server objects.

sync:///Projects[/ProjectName]/[pathtoObject]
Controls the permission for the specified project
or the entire project area on the server.

sync:///Modules[/Category [...]][/ModName]
Controls the permission for the specified module,
category, or entire module area on the server.

Notes:
You can use wildcards, such as * to specify all
matching objects within the specified path, for
example: sync:///Modules/Chip/*.c controls access
to all .c files within the module Chip. If Chip
was a category and you wanted to specify all
modules within the category, you could specify
the URL either of the following ways:
sync:///Modules/Chip/*
sync:///Modules/Chip

 The sync:/// and sync:///* URLs are generic Sync
URLs that can be used to provide default access
for the server.
```

RETURN VALUE

```
Does not return a TCL value. When successful, this command lists the
categories and permissions for the object, alphabetically by
category. If there is an error, DesignSync return an appropriate
error message explaining the failure.
```

SEE ALSO

```
acadmin addobj, acadmin rmobj, acadmin setcatperm
```

EXAMPLES

- Example Showing the List of Permissions for an Object in ACAdmin

Example Showing the List of Permissions for an Object in ACAdmin

```
This example shows the list of permissions for a specified object in
```

```
ACAdmin.
dss> acadmin listperms -object sync:///Modules/ChipDev/ALU -server \
sync://serv1.ABCo.com:2746

 DS-READ {ALL {}}
 DS-PROJADMIN {ALL {}}
 ADMIN-MODULE {ALL {anewman rsmith}}
 BROWSE {ALL {}}
 ACAProjectDefs {ALL {}}
 DS-WRITE {ALL {}}
```

acadmin listusers

acadmin listusers Command

NAME

```
acadmin listusers   - lists users for the server or specified object
```

DESCRIPTION

```
The acadmin listusers command lists users associated with acadmin
objects or groups. As shown in the example below, user-defined groups
are expanded in the output, so you will not see the user-defined
group name.  Virtual groups, such as All-Module-Owners, are not
expanded and you may see them referenced in the command output. For
more information on virtual groups and how to use them, see the
ENOVIA Synchronicity Access Control Guide.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

To use this command, the Access Control Administrator must be enabled
on the server being updated.
```

SYNOPSIS

```
acadmin listusers [-object <ObjectURL>] [-server <SyncURL>]
```

ARGUMENTS

```
-server <SyncURL>   Specifies the option Sync URL as follows:
                    sync://<host>[:<port>] or
                    syncs://<host>[:<port>]
                    where 'sync://' or 'syncs://' are required,
                    <host> is the machine on which the SyncServer is
                    installed, and <port> is the SyncServer port
                    number (defaults to 2647/2679).  For example:
```

```
                    sync://apollo.spaceco.com:1024

                    If this argument is not specified, DesignSync
                    users the environment variable SYNC_SERVER_URL.

                    If there is no SYNC_SERVER_URL variable defined,
                    DesignSync uses the vault association set on the
                    current working directory. If there is no vault
                    or there are multiple vaults associated with the
                    directory, for example, when you have overlapping
                    modules, the command fails.
```

OPTIONS

- -object

-object

```
 -object <objectURL> Enter the Sync URL of the object on which to view
                     the users. You can view the users for any object
                     on the server. You may specify objects as
                     follows:

                     sync:///
                     Controls the default permissions for all
                    object-independent actions on the server.  For
                    example, user creation and removal actions do not
                    depend on specific server objects. (Default)

                     sync:///*
                     Controls the default permissions for all
                     object-dependent actions on the server. For
                     example, browsing objects on the server, creating
                     or modifying modules, etc.  depend on having
                     access to related server objects.

                     sync:///Projects[/ProjectName]/[pathtoObject]
                     Controls the permission for the specified project
                     or the entire project area on the server.

                     sync:///Modules[/Category [...]][/ModName]
                     Controls the permission for the specified module,
                     category, or entire module area on the server.

                     Notes:
                     You can use wildcards, such as * to specify all
                     matching objects within the specified path, for
                     example: sync:///Modules/Chip/*.c controls access
                     to all .c files within the module Chip. If Chip
                     was a category and you wanted to specify all
                     modules within the category, you could specify
                     the URL either of the following ways:
                     sync:///Modules/Chip/*
```

```
                    sync:///Modules/Chip

                    The sync:/// and sync:///* URLs are generic Sync
                   URLs that can be used to provide default access
                   for the server.
```

RETURN VALUE

```
This command doesn't return any TCL values. If the command is
successful, you will see a list of users who have been assigned to
acadmin categories. If the command fails, it will return an
appropriate error message explaining the failure.
```

SEE ALSO

```
acadmin addusers, acadmin rmusers, acadmin listgroups
```

EXAMPLES

- Example of Listing Users Associated with Categories on the Server

Example of Listing Users Associated with Categories on the Server

```
This example shows the users associated with acadmin categories on
the server. This is a very simple example which has very open
permissions except for one category assigned to the user group
DocWriters. The user rsmith is the only user in the group
DocWriters, therefore, his is the only username that appears
separately.

dss> acadmin listusers -server sync://serv1.ABCo.com:2476

 everyone rsmith
```

acadmin reset

acadmin reset Command

NAME

```
acadmin reset       - Regenerates the AccessControl file and runs an
                      Access Reset.
```

DESCRIPTION

The acadmin reset command regenerates the AccessControl to collect all the changes made to the system since the last reset. Then the command runs the Access Reset to load the changes onto the server.

SYNOPSIS

```
acadmin reset [-server <SyncURL>]
```

ARGUMENTS

-server <SyncURL>	Specifies the option Sync URL as follows: sync://<host>[:<port>] or syncs://<host>[:<port>] where 'sync://' or 'syncs://' are required, <host> is the machine on which the SyncServer is installed, and <port> is the SyncServer port number (defaults to 2647/2679). For example: sync://apollo.spaceco.com:1024 If this argument is not specified, DesignSync users the environment variable SYNC_SERVER_URL. If there is no SYNC_SERVER_URL variable defined, DesignSync uses the vault association set on the current working directory. If there is no vault or there are multiple vaults associated with the directory, for example, when you have overlapping modules, the command fails.

RETURN VALUE

This command does not return a TCL value. When successful, the command displays a success result. When it fails, the command displays an error explaining the failure.

SEE ALSO

acadmin addgroup, acadmin addgroupusers, acadmin addobj, acadmin addusers, acadmin rmgroup, acadmin rmgroupusers, acadmin rmobj, acadmin rmusers, acadmin setcatperm

EXAMPLES

- ACAdmin Reset Example

ACAdmin Reset Example

```
This example shows the response you see when an ACAdmin Reset is
successful.

dss> acadmin reset -server sync://serv1.ABCo.com:2647

 Updated AccessControl file
 Reset Access Control
```

acadmin rmgroup

acadmin rmgroup Command

NAME

```
acadmin rmgroup     - Remove a user group.
```

DESCRIPTION

```
This command removes user groups. These user groups, when removed, are
removed from all the command categories they were associated with.

Notes:
 o Usually only the DesignSync administrators should have permission
   to remove or modify user group definitions.

 o There are three dynamic (or virtual) user groups that can never be
   removed. They are All-Module-Owners, All-Project-Owners, and
   All-Server-Users. These groups are automatically generated when
   ACAdmin is reset.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

To use this command, the Access Control Administrator must be enabled
on the server being updated.
```

SYNOPSIS

```
acadmin rmgroup -group <groupName> [-server <SyncURL>]
```

ARGUMENTS

- Sync URL

Sync URL

```
-server <SyncURL>   Specifies the option Sync URL as follows:
                    sync://<host>[:<port>] or
                    syncs://<host>[:<port>]
                    where 'sync://' or 'syncs://' are required,
                    <host> is the machine on which the SyncServer is
                    installed, and <port> is the SyncServer port
                    number (defaults to 2647/2679).  For example:
                    sync://apollo.spaceco.com:1024

                    If this argument is not specified, DesignSync
                    users the environment variable SYNC_SERVER_URL.

                    If there is no SYNC_SERVER_URL variable defined,
                    DesignSync uses the vault association set on the
                    current working directory. If there is no vault
                    or there are multiple vaults associated with the
                    directory, for example, when you have overlapping
                    modules, the command fails.
```

OPTIONS

```
-group <groupName>  Name of the user group. This name should
                    match exactly the name used when the group was
                   added, including case. If you are not sure how to
                   correctly specify the name of the group being
                   removed, you can list the available groups using
                   the acadmin listgroup command.
```

RETURN VALUE

```
This command doesn't return any TCL values. If the command succeeds,
you'll receive a success message.  If the command fails, you'll
receive an error message explaining the failure.
```

SEE ALSO

```
acadmin addgroup, acadmin listgroups, acadmin rmgroupusers
```

EXAMPLES

Example of Removing a Group

```
This example shows removing a group from the server.

dss> acadmin rmgroup -group DocWriters -server \
```

```
sync://serv1.ABCo.com:2647

 Deleted 1 User Group(s)
 Updated Permissions file
```

acadmin rmgroupusers

acadmin rmgroupusers Command

NAME

```
acadmin rmgroupusers- Remove users from group.
```

DESCRIPTION

```
This command removes defined users from an existing group.  Both
usernames and groupsnames are case sensitive. You can only remove
users from one group at a time.

This does not remove users or groups from the system. In order to
delete users or groups, you must use the acadmin rmusers or acadmin
rmgroups command respectfully.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

To use this command, the Access Control Administrator must be enabled
on the server being updated.

To apply the changes created by this command to the server, you must
reset the server using the acadmin reset command or ACAdmin Reset
from the DesignSync Web UI.
```

SYNOPSIS

```
acadmin rmgroupusers -group <groupName> -users <userlist>
                     [-server <SyncURL>]
```

ARGUMENTS

- Sync URL

Sync URL

```
-server <SyncURL>   Specifies the option Sync URL as follows:
                    sync://<host>[:<port>] or
                    syncs://<host>[:<port>]
```

```
where 'sync://' or 'syncs://' are required,
<host> is the machine on which the SyncServer is
installed, and <port> is the SyncServer port
number (defaults to 2647/2679).  For example:
sync://apollo.spaceco.com:1024

If this argument is not specified, DesignSync
users the environment variable SYNC_SERVER_URL.

If there is no SYNC_SERVER_URL variable defined,
DesignSync uses the vault association set on the
current working directory. If there is no vault
or there are multiple vaults associated with the
directory, for example, when you have overlapping
modules, the command fails.
```

OPTIONS

-group

```
-group <groupName>  Name of the group to which the users are being
                    removed. Group names are case sensitive.

                    If you are not sure what groups exist, use the
                    acadmin listgroups command to get a list of
                    existing groups.

                   Note: If the group is associated with an object,
                    you need to specify the fully extended group
                    name.  Fully extended group name is specified as:

                    <GroupName>@sync:///<ServerObjectPath>
                   Where <GroupName> is the case sensitive name of
                    the group.
                   <ServerObjectPath> is the path to the object on
                    which the group was created for example:
                   Projects/[<ProjectFolder>/...]ProjectName
                   Modules/[<Category>/...]ModuleName
```

-user

```
-user <userlist>    A comma separated list of users to associate with
                    the group. User names are case sensitive. If you
                   are not sure what usernames to specify, user the
                   acadmin listusers command to get a list of
                   existing users. You may also use the special user
                   definitions provided by DesignSync.  For more
                   information on the special users, see the Access
```

```
                    Administration Guide.
```

RETURN VALUE

```
This command does not return any TCL values.  When successful, the
command indicates that the usergroup was removed. If the command
fails, DesignSync returns an appropriate error message explaining the
failure.
```

SEE ALSO

```
acadmin addgroup, acadmin addusers, acadmin addgroupusers,
acadmin listusers, acadmin listgroups, acadmin rmgroup
```

EXAMPLES

- Example of Removing a User from a Group

Example of Removing a User from a Group

```
This example shows removing a user from an acadmin group.

dss> acadmin rmgroupusers -user rsmith -group DocWriters -server \
sync://serv1.ABCo.com:2647

Updated 1 User Group(s)
```

acadmin rmobj

acadmin rmobj Command

NAME

```
acadmin rmobj       - Remove the object from ACAdmin management
```

DESCRIPTION

```
This command removes all the categories defined for the specified
object from the acadmin configuration files. The object must have
been added to acadmin to manage, but, the object does not need to
exist on the server.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

To use this command, the Access Control Administrator must be enabled
on the server being updated.
```

To apply the changes created by this command to the server, you must reset the server using the acadmin reset command or ACAdmin Reset from the DesignSync Web UI.

SYNOPSIS

```
acadmin rmobj -object <objectURL> [-server <SyncURL>]
```

ARGUMENTS

`-server <SyncURL>` Specifies the option Sync URL as follows: sync://<host>[:<port>] or syncs://<host>[:<port>] where 'sync://' or 'syncs://' are required, <host> is the machine on which the SyncServer is installed, and <port> is the SyncServer port number (defaults to 2647/2679). For example: sync://apollo.spaceco.com:1024

If this argument is not specified, DesignSync users the environment variable SYNC_SERVER_URL.

If there is no SYNC_SERVER_URL variable defined, DesignSync uses the vault association set on the current working directory. If there is no vault or there are multiple vaults associated with the directory, for example, when you have overlapping modules, the command fails.

OPTIONS

`-object <objectURL>` Enter the Sync URL of the object to remove from ACAdmin control. You may specify objects as follows:

sync:///Projects[/ProjectName]/[pathtoObject] Controls the permission for the specified project or the entire project area on the server.

sync:///Modules[/Category [...]][/ModName] Controls the permission for the specified module, category, or entire module area on the server.

Note: The sync:/// and sync:///* URLs are generic Sync URLs that can be used to provide default access for the server.

RETURN VALUE

```
This command does not return any TCL values. When successful, the
command returns a message indicating that the object was
removed. If the command fails, it returns an error message explaining
the failure.
```

SEE ALSO

```
acadmin addobj, acadmin rmgroup, acadmin rmgroupusers
```

EXAMPLES

- Example of Removing an Object from ACAdmin Management

Example of Removing an Object from ACAdmin Management

```
This example shows removing an object from ACAdmin management.

dss> acadmin rmobj -object sync:///Modules/ChipDev/ALU -server \
sync://server1.ABCo.com:2647

 Object sync:///Modules/ChipDev/ALU has been deleted from AC
 definitions.
```

acadmin rmusers

acadmin rmusers Command

NAME

```
acadmin rmusers     - Removes a user(s) from the specified Object and
                      Category.
```

DESCRIPTION

```
This command removes one or more users from a specified category of
permissions assigned to an object. This allows you to modify the list
of users assigned to a permissions category as needed.

users are assigned to a permissions category as your needs change.

 Note: The user does not need to exist when the command is run, as
 long as it is present in the acadmin configurations files. For more
 information on creating users in DesignSync see the DesignSync
 Administrator's Guide.
```

```
This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

To use this command, the Access Control Administrator must be enabled
on the server being updated.

To apply the changes created by this command to the server, you must
reset the server using the acadmin reset command or ACAdmin Reset
from the DesignSync Web UI.
```

SYNOPSIS

```
acadmin rmusers -category <categoryName> -object <objectURL>
                -users <userlist> [-server <SyncURL>]
```

ARGUMENTS

- Sync URL

Sync URL

```
-server <SyncURL>   Specifies the option Sync URL as follows:
                    sync://<host>[:<port>] or
                    syncs://<host>[:<port>]
                    where 'sync://' or 'syncs://' are required,
                    <host> is the machine on which the SyncServer is
                    installed, and <port> is the SyncServer port
                    number (defaults to 2647/2679).  For example:
                    sync://apollo.spaceco.com:1024

                    If this argument is not specified, DesignSync
                    users the environment variable SYNC_SERVER_URL.

                    If there is no SYNC_SERVER_URL variable defined,
                    DesignSync uses the vault association set on the
                    current working directory. If there is no vault
                    or there are multiple vaults associated with the
                    directory, for example, when you have overlapping
                    modules, the command fails.
```

OPTIONS

- -category
- -object
- -user

-category

```
-category           Name of the command category associated with the
<categoryName>      user.
```

-object

```
-object <objectURL> Sync URL of the object associated with the user.
```

-user

```
-user <userlist>    A comma separated list of users to remove from
                    the category and object.
```

RETURN VALUE

```
This command does not return a TCL value. The command displays a
success message when successful or an appropriate error message if the
command fails.
```

SEE ALSO

```
acadmin addobj, acadmin addgroup, acadmin addgroupusers
```

EXAMPLES

- Example of Removing a User from a Category on an Object

Example of Removing a User from a Category on an Object

```
This example shows the removal of a user from a category defined for
an object.

dss> acadmin rmusers -object sync:///Modules/Chip/ALU -category \
     ADMIN-MODULE -user rsmith -server sync://serv1.ABCo.com:2647

 Category ADMIN-MODULE updated
```

acadmin setcatperm

acadmin setcatperm Command

NAME

```
acadmin setcatperm  - Set permission for users, objects, & categories
```

DESCRIPTION

```
This command allows you to adjust the permissions associated with
categories, and optionally users, associated with objects managed by
ACAdmin.

Before you can adjust the permissions, you need to associate the
objects with ACAdmin using the acadmin addobj command. This also
makes the defined categories available for the object with the
permissions specified by the command.

Once those have been created, the permissions for the category can be
adjusted for all users or individual users or user groups assigned to
the category.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

To use this command, the Access Control Administrator must be enabled
on the server being updated.

To apply the changes created by this command to the server, you must
reset the server using the acadmin reset command or ACAdmin Reset
from the DesignSync Web UI.
```

SYNOPSIS

```
acadmin setcatperm -category <categoryName> -object <objectURL>
                   -permission <permission> [-users <userlist>]
                   [-server sync(s)://host:port]
```

ARGUMENTS

- Sync URL

Sync URL

```
-server <SyncURL>   Specifies the option Sync URL as follows:
                    sync://<host>[:<port>] or
                    syncs://<host>[:<port>]
                    where 'sync://' or 'syncs://' are required,
                    <host> is the machine on which the SyncServer is
                    installed, and <port> is the SyncServer port
                    number (defaults to 2647/2679).  For example:
                    sync://apollo.spaceco.com:1024

                    If this argument is not specified, DesignSync
```

```
                    users the environment variable SYNC_SERVER_URL.

                    If there is no SYNC_SERVER_URL variable defined,
                    DesignSync uses the vault association set on the
                    current working directory. If there is no vault
                    or there are multiple vaults associated with the
                    directory, for example, when you have overlapping
                    modules, the command fails.
```

OPTIONS

- -category
- -object
- -permission
- -user

-category

```
 -category           Name of the command category being modified.
 <categoryName>      The command category must exist already.
```

-object

```
 -object <objectURL> Enter the Sync URL of the object on which to set
                     the permissions. You can set the permissions for
                     any object on the server. You may specify objects
                     as follows:

                     sync:///
                     Controls the default permissions for all
                    object-independent actions on the server. For
                    example, user creation and removal actions do not
                    depend on specific server objects. (Default)

                     sync:///*
                     Controls the default permissions for all
                     object-dependent actions on the server.  For
                     example, browsing objects on the server, creating
                     or modifying modules, etc.  depend on having
                     access to related server objects.

                     sync:///Projects[/ProjectName]/[pathtoObject]
                     Controls the permission for the specified project
                     or the entire project area on the server.

                     sync:///Modules[/Category [...]][/ModName]
                     Controls the permission for the specified module,
                     category, or entire module area on the server.

                     Note:
                     The sync:/// and sync:///* URLs are generic Sync
```

URLs that can be used to provide default access
for the server.

-permission

-permission <permission>
Enter a comma separated list of defined permissions to associate with the object.

ALL, LIST, EXCLLIST, NONE, SERVDEF

When LIST or EXCLLIST are used, you can specify a user or userlist for whom to set the permissions.

-user

-user <userlist>
A comma separated list of users whose permissions are being modified for the specified category.

If no users are specified, the permissions for all users in the category are modified.

RETURN VALUE

This command does not return any TCL values. When successful, lists the categories that have been modified, or returns an appropriate error message if the command fails.

SEE ALSO

acadmin addgroup, acadmin addobj, acadmin addusers,
acadmin listcats, acadmin listobjs, acadmin listperms, acadmin reset

EXAMPLES

- Example of Setting Permissions for a Category
- Example of Setting Permissions for a User

Example of Setting Permissions for a Category

This example shows how to set permissions for a category defined for a module object. The acadmin listperms command following shows

```
the permissions for the category.

dss> acadmin setcatperm -object sync:///Modules/Chip/ALU -category \
ADMIN-MODULE -permission ALL -server sync://serv1.ABCo.com:2647

 Category ADMIN-MODULE updated

dss> acadmin listperms -object sync:///Modules/Chip/ALU -category \
ADMIN-MODULE -server sync://serv1.ABCo.com:2647

 ALL rsmith
```

Example of Setting Permissions for a User

```
This example shows how to set permissions for a user associated with
a category defined for a module object.
dss> acadmin setcatperm -object sync:///Modules/Chip/ALU -category \
 ADMIN-MODULE -permission LIST -user rsmith -server \
 sync://lwvrh17mon:30126

  Category ADMIN-MODULE updated

dss> acadmin listperms -object sync:///Modules/Chip/ALU -category \
ADMIN-MODULE -server sync://lwvrh17mon:30126

 LIST rsmith
```

Access Control Commands

access Commands

NAME

```
access              - Access-control commands
```

DESCRIPTION

- Notes for Modules (Module-based)

```
These commands provide access to the access control system used by
DesignSync tools. Note that some access control commands (access
allow, access define, access deny, access filter,
access global, access init) are available ONLY within an
AccessControl file. See the ENOVIA Synchronicity Access Control Guide
for more information.
```

Notes for Modules (Module-based)

```
Note: The access decline command is only available for modules access
controls and is used only within an access control file.
```

SYNOPSIS

```
access <access_command> [<access_command_options>]

Usage: access [allow|db_filter|decline|define|deny|filter|global|
               init|list|reset|verify]
```

OPTIONS

```
Vary by command.
```

RETURN VALUE

```
Varies by command.
```

SEE ALSO

```
stcl, server-side, rstcl, access reset, access verify
```

EXAMPLES

```
See specific "access" commands.
```

access

access Commands

NAME

```
access              - Access-control commands
```

DESCRIPTION

- Notes for Modules (Module-based)

```
These commands provide access to the access control system used by
DesignSync tools. Note that some access control commands (access
allow, access define, access deny, access filter,
access global, access init) are available ONLY within an
AccessControl file. See the ENOVIA Synchronicity Access Control Guide
for more information.
```

Notes for Modules (Module-based)

```
Note: The access decline command is only available for modules access
controls and is used only within an access control file.
```

SYNOPSIS

```
access <access_command> [<access_command_options>]

Usage: access [allow|db_filter|decline|define|deny|filter|global|
               init|list|reset|verify]
```

OPTIONS

```
Vary by command.
```

RETURN VALUE

```
Varies by command.
```

SEE ALSO

```
stcl, server-side, rstcl, access reset, access verify
```

EXAMPLES

```
See specific "access" commands.
```

access allow

access allow Command

NAME

```
access allow        - Allows access to the specified actions
```

DESCRIPTION

```
The 'access allow' and related 'access deny' commands allow or deny
access to a specified list of actions. You can allow or deny access
for particular users and under circumstances you specify.

See the ENOVIA Synchronicity Access Control Guide for details on
setting up and using access controls.
```

SYNOPSIS

```
access {allow | deny} <actionList> {everyone | [only] users <userList>}
       [when <parm> <globExpr> [when ...]]
       [-because "<message_string>"]
```

OPTIONS

- actionList
- everyone
- users
- userList
- when
- -because

```
Note: For list parameters, (<actionList> and <userList>) surround
multiple values with braces, for example, {Checkin Checkout}.
The braces are optional when you specify a single value; for example,
both {Checkout} and Checkout are valid.
```

actionList

```
actionList      The name of the actions to be controlled. The default
                actions are defined in the default Synchronicity
                access control file by 'access define' statements
                and have names such as Checkin, Checkout,
                and Delete. Note: Actions are case sensitive.
```

everyone

```
everyone        Specifies that the access control applies to all users.
```

users

users	Specifies the group of users to which the access control applies. The 'only' modifier to the users argument indicates that users not on the list are assigned opposite access permissions. For example, 'access allow Checkin users Joe' means Joe is allowed to check in files, but 'access allow Checkin only users Joe' means that nobody other than Joe is allowed to check in files. One use of 'only' is to define restrictive access rights (deny access to everyone) and then specifically grant rights to certain users using the 'only' modifier. Users can be named more than once by multiple allow and deny commands. Access by users listed as both allowed and denied is determined by the last list in which they appear.

userList

userList	The list of users to which 'access allow' or 'access deny' applies. Surround a multiple list of users with braces; the braces are optional for a single user. Note: User names are case sensitive.

when

when	Use optional 'when' clauses to indicate that the <userList> is only allowed or denied access when the named parameter <parm> (corresponding to the <parameterList> argument of the 'access define' command) matches the glob-style expression given by <globExpr>. If multiple 'when' clauses are used, all of them must match in order for the access rights to be affected; in other words, 'when' clauses are joined with an implicit AND operator.

-because

-because	Use optional -because clauses with 'access allow' and 'access deny' statements to provide a message string to users indicating why an action failed. In 'access allow' statements, use the -because clause with the 'only' modifier to explain under which circumstances the 'only' modifier is restricting access. If an AccessControl file contains multiple 'access allow', 'access deny', or 'access filter' statements for an action, the -because clause of the last 'access allow|deny' statement

(or the return message string in the case of an 'access filter' statement) is returned if the action fails. If a -because or return message string is not included in the last 'access allow', 'access deny', or 'access filter' statement, only the default "Permission denied by the AccessControl system" message is returned.

RETURN VALUE

none

SEE ALSO

access decline, access filter, access reset, access verify

EXAMPLES

The following skeleton example uses the wildcard character passed to access verify to ensure that notes for which users have only partial view access are not entirely suppressed from the GUI:

```
access allow ViewNote everyone when id \\*
```

You also can use constructs of this type for EditNote and DeleteNote actions.

For additional examples of using 'access allow' and 'access deny,' see the ENOVIA Synchronicity Access Control Guide.

access db_filter

access db_filter Command

NAME

access db_filter - Specify criteria for allowing or denying access for ViewNote or EditNote

DESCRIPTION

This command lets you specify criteria for allowing or denying access to users attempting to view or edit notes. The access db_filter command can be used only for ViewNote and EditNote filters. For any ViewNote or EditNote action that requires verifying more than one note, access db_filter performs better than access filter.

The access db_filter command is used only in scripts in conjunction with the note query -filter command.

The access db_filter rule always gets a parameter query. When this rule is invoked by the note query -filter command, this parameter contains an unfiltered query string. The filter script can use this unfiltered query in any way. Usually the filter script constructs a complex query based on the unfiltered query. This complex query then expresses access control constraints. In this way, access control verification is done at once for all the notes resulting from the initial query.

An access db_filter script must use the directives ALLOW, DENY, ALLOW_ALL, or DENY_ALL. It must not return any value.

Most access rules (access allow, access deny, access filter) operate on one note at a time and can return only a single value (ALLOW, DENY or UNKNOWN).

However, an access db_filter rule returns results in a different way from other rules because it can operate on multiple notes gathered from the note query command. Instead of returning a tri-state value, it modifies a tri-state value (0 (denied), 1 (allowed), ? (unknown)) in a Tcl array. The access commands use this Tcl array to determine which notes are passed back to the user through the note query -filter command.

Before entering an access db_filter block, all notes for which access is to be determined are stored in the ACCESS array. Each array value is initially set to "?", indicating that access has not been determined.

When an unfiltered query is passed to access db_filter, it is allowed to modify the array ACCESS as appropriate by the supplied API. When access is granted, the array element changes to 1; when access is denied, the element changes to 0. At the end of the rule evaluation, the array ACCESS is checked for the presence of unknown elements. If all elements are in a known state, the loop over the rules is aborted. If, for instance, the last rule in a ViewNote action list is:

access allow NoteActions masteradmin

this rule is the only one processed for the masteradmin user.

An access db_filter rule never adds entries to the array ACCESS; it can only modify existing unknown entries (containing "?") and can never set a value to ?.

When access db_filter is called, the unfiltered query has already been executed, extracting only ID's. The access db_filter can:

-Rerun the query, tacking on a filter expression, using FILTERED_IDS.

-Run a related query, and then do its own "join".

```
When writing custom access db_filter scripts, do not directly access
the ACCESS array. Instead, use the functions provided by the API,
listed in the API FUNCTIONS section.

The access db_filter rule can coexist with other rules, such as access
allow or access deny. All combinations of the note query -filter and
access verify commands with access allow, access deny, access filter,
and access db_filter rules are valid. Different rules for the same
action are applied in the same order as in the current system, with the
last rule providing a definite answer.

You add 'access db_filter' commands to the site or server
AccessControl file within the <SYNC_CUSTOM_DIR> hierarchy (defaults
to <SYNC_DIR>/custom):

Site-wide:
    <SYNC_SITE_CUSTOM>/share/AccessControl
    (where <SYNC_SITE_CUSTOM> defaults to <SYNC_CUSTOM_DIR>/site)
Server-specific (UNIX only):
    <SYNC_CUSTOM_DIR>/servers/<host>/<port>/share/AccessControl

Note: Do not edit any of the access control files in the
$SYNC_DIR/share area; you should edit the site or server AccessControl
file. See the ENOVIA Synchronicity Access Control Guide for details on
setting up and using access controls.
```

SYNOPSIS

```
access db_filter <actionList> [when <parm> <globExpr> [when ...]]
                 <script>
```

OPTIONS

- actionList
- when
- script

```
Note: For the list parameter <actionList>, surround multiple values
with braces, for example, {ViewNote EditNote}. The braces are optional
when you specify a single value; for example, both {ViewNote} and
EditNote are valid.
```

actionList

```
actionList      The name of the actions to be controlled -
                ViewNote and/or EditNote. These actions are defined
                in the default Synchronicity access control
                file by 'access define' statements
```

when

when	Use optional 'when' clauses to indicate that the user is only allowed or denied access when the named parameter <parm> (corresponding to the <parameterList> argument of the 'access define' command) matches the glob-style expression given by <globExpr>. If multiple 'when' clauses are used, all of them must match for the access rights to be affected. In other words, 'when' clauses are ANDed.

script

script	A Tcl script or Tcl statements supplied directly to the 'access db_filter' command and evaluated to determine whether a given user should be allowed or denied access to the named action for each note returned by the query. When invoked, <script> is passed a parameter named $user that holds the name of the user whose access is in question, in addition to the parameters listed in the access define statement that defined the action. Another parameter available to the script is $action, which always has either the value EditNote or ViewNote, depending on what action is being verified. Using the $action parameter, you can invoke different commands depending on which action has triggered the filter statement. An access db_filter script must use the directives ALLOW, DENY, ALLOW_ALL, or DENY_ALL.

API FUNCTIONS

- ALLOW
- ALLOW_ALL
- CHECK_STAR
- DENY
- DENY_ALL
- FILTERED_IDS
- FOREACH_NOTE
- FOREACH_UNKNOWN
- SINGLE_NOTE

The access db_filter contains the following API functions for accessing the ACCESS array:

ALLOW

ALLOW - This function grants access to a single note and sets the ACCESS array element for that note to "1". For example:

```
ALLOW $id
```

ALLOW_ALL

ALLOW_ALL - This function sets all notes as allowed--that is, access is granted. All ACCESS array elements are set to "1". For example:

```
ALLOW_ALL
```

CHECK_STAR

CHECK_STAR - This function is used when access is always granted because an asterisk is passed to the access verify command. In this case, the CHECK_STAR function grants access and immediately returns to the calling application code. You therefore do not need to return after calling CHECK_STAR. This function should appear at the top of every access db_filter call unless the asterisk case is handled differently.

See the EXAMPLES section for an example of using this function.

DENY

DENY - This function denies access to a single note and sets the ACCESS array element for that note to "0". For example:

```
DENY $id
```

DENY_ALL

DENY_ALL - This function sets all notes as denied--that is, access is denied. All ACCESS array elements are set to "0". For example:

```
DENY_ALL
```

FILTERED_IDS

FILTERED_IDS - This function returns a filtered list of note IDs based on a set of specified criteria. The variables listed below are available to every access db_filter script. For example:

```
# Other users can access their own notes.
set AC_squery "[sq $user] = f_Author"

set AC_notes [FILTERED_IDS $type $sqlquery $dbquery
$attached $AC_squery <$selectList>]
```

where:

- $type is the note type passed into the filter.
- $sqlquery is the original SQL query passed in to the filter, if any.
- $dbquery is the original database query passed in to the filter, if any.
- $attached is the URL for note query -attached calls passed in to the filter, if any.
- $AC_squery is an additional SQL query describing the additional criteria needed to determine access on the set of notes.
- $selectList is an optional parameter that allows you to specify extra properties from the note type that the filter needs to determine access. If you specify $selectList, it returns a list of lists in the form: {id prop prop ...} {id prop prop ...}.

FOREACH_NOTE

FOREACH_NOTE - This function iterates over the ACCESS array and returns one note at a time, regardless of whether the note has been allowed or denied by a previous ALLOW, DENY, ALLOW_ALL or DENY_ALL command. For example:

```
FOREACH_NOTE <varName> {
  # Process the note, whose ID is stored in <varName>
  ALLOW $<varName>
}
```

FOREACH_UNKNOWN

```
FOREACH_UNKNOWN -This function iterates over the ACCESS array and
                 returns one note at a time.  Only notes marked as
                 unknown ("?") are returned. For example:

                 FOREACH_UNKNOWN <varName> {
                   # Process the note, whose ID is stored in <varName>
                   DENY $<varName>
                 }
```

SINGLE_NOTE

```
SINGLE_NOTE -    This function checks whether a single note is being
                 access controlled, and returns the note ID for a
                 single note or a null string if there are multiple
                 notes.

                 It is not necessary for access db_filter to know
                 whether a single note or multiple notes are being
                 checked; the filter must be able to handle 0, 1, or
                 multiple notes. However, in some instances, an
                 optimization is possible if you know that a single
                 note is being checked.

                 For example:

                 set noteId [SINGLE_NOTE]
                 if {$noteId == ""} {
                   # Multiple notes are being access controlled
                 } else {
                   # Only one note, $noteId contains the note id
                 }
```

RETURN VALUE

```
none
```

SEE ALSO

```
access filter, access global, access init, access reset, access verify
```

EXAMPLES

- Set Different Access Rights for a Note
- Create a Filter to View Notes Authored By User

Set Different Access Rights for a Note

```
In this example, the script passed to 'access db_filter' ensures that
SecretIngredient notes can be viewed by anyone if their secrecy level
is low; otherwise, only their Inventor can view these notes. But
SecretIngredient notes can be edited only by their Inventor,
regardless of the secrecy level.

The script sets up a query, filter_squery, an SQL query that gathers
up all the notes whose Inventor is the current user if the action is
EditNote.  The query gathers up both the notes whose Inventor is the
current user and the notes with Secrecy level set to 'low' if the
action is ViewNote. Note that in SQL queries, you must prepend 'f_'
to the property name, for example, 'f_Inventor' and 'f_Secrecy'.
The query is passed to the FILTERED_IDS function which actually
gathers the notes.  Finally, for all the notes that match the query,
the EditNote or ViewNote action is set to ALLOW.

access db_filter {ViewNote EditNote}  when type SecretIngredient {
   CHECK_STAR
   # CHECK_STAR checks if first note Id == '*' and
   # allows action if so; this is used so that the
   # note type shows up in the Quick View panel.

   if {$action == "ViewNote"} {
      set filter_squery "f_Inventor = [sq $user] OR f_Secrecy = 'low'"
   } else {
      set filter_squery "f_Inventor = [sq $user]"
   }
   set filtered_notes [FILTERED_IDS $type $sqlquery $dbquery \
                       $attached $filter_squery]
   # The FILTERED_IDS function runs a subquery, which
   # is the original query with the filter_squery tacked
   # onto it.



   # filtered_notes now holds a list of notes that matches the
   # original query,
   # AND passed the access checks. Now update the ACCESS map
   # by setting the flag for each note that passed muster and
   # denying those that didn't

   foreach AC_note $filtered_notes {
     ALLOW $AC_note
   }

   FOREACH_UNKNOWN AC_note {
     DENY $AC_note
   }
}
```

Create a Filter to View Notes Authored By User

This example creates a filter to display notes authored by the user who runs the query.

```
access db_filter ViewNote when type "SyncDefect" {

   # ---------------------------------------------------------------
   # The following variables are available to this filter:
   #
   #     $type . . . . .  The note type
   #     $sqlquery . . .  The original SQL query, passed in from
   #                      the note query command, if any.
   #     $dbquery  . . .  The original DB query, passed in from the
   #                      the note query command, if any.
   #     $attached . . .  A note attachment, passed in from the note
   #                      query -attached command, if any.
   #     $user . . . . .  The user ID of the user requesting access.
   # ---------------------------------------------------------------

   # Check for the "*" operator, which could be sent in from a
   # note panel.  If a "*" is found, the filter will return
   # immediately after granting access.

   CHECK_STAR

   # For purposes of this example filter, allow users to view
   # only the SyncDefect notes that they authored.  Therefore,
   # build a new SQL query to gather only those notes where the
   # user is the author.

   Set AC_squery "[sq $user] = f_Author "

   # Gather the filtered list of note IDs where the criteria
   # matches the original query and our criteria.

   set AC_ids [FILTERED_IDS \
       $type $sqlquery $dbquery $attached $AC_squery]

   # The variable AC_ids now holds the list of note IDs that
   # pass the original query (say, we queried for all "open" defects)
   # AND match our criteria of the user being the author.  These are
   # the notes the user has access to view. Therefore, we
   # iterate through the ID list and allow each note returned from
   # FILTERED_IDS.

   foreach noteId $AC_ids {
       ALLOW $noteId
   }

   # Now DENY all the notes that did NOT match our criteria.
   # Iterate through all the notes that are still in an UNKNOWN
   # state and DENY them.

   FOREACH_UNKNOWN noteId {
       DENY $noteId
   }
}
```

access decline

access decline Command

NAME

```
access decline      - Causes additional access rules to be invoked
```

DESCRIPTION

```
The "access decline" command is used with module data. When you
operate on a module, you are operating on the module as well as on
the module's individual members. Access control rules can apply at
the module level, and also at the individual module member level.

An operation on module data will first check module level access. If the
module level access is allowed or denied, then no further checks are
necessary. However, if the module level access is declined, then an access
check is needed, for each individual module member that is participating
in the operation.

The "access decline" command declines access to a specified list of
actions. You can also decline access for particular users and under
circumstances you specify.

Attempting to decline access for a command that does not support the
DECLINE outcome results in an outcome of DENY.

See the ENOVIA Synchronicity Access Control Guide for details on
module access, setting up, and using access controls.
```

SYNOPSIS

```
access decline <actionList> {everyone | users <userList>}
               [when <parm> <globExpr> [when ...]]
```

OPTIONS

- actionList
- everyone
- users
- userList
- where

```
Note: For list parameters, (<actionList> and <userList>) surround
multiple values with braces, for example, {Checkin Checkout}.
The braces are optional when you specify a single value; for example,
```

both {Checkout} and Checkout are valid.

actionList

actionList	The name of the actions to be controlled. The default actions are defined in the default Synchronicity AccessControl file by "access define" statements and have names such as Checkin and Checkout. Note: Actions are case sensitive.

everyone

everyone	Specifies that the access control applies to all users.

users

users	Specifies the group of users to which the access control applies.

userList

userList	The list of users to which "access decline" applies. Surround a multiple list of users with braces; the braces are optional for a single user. Note: User names are case sensitive.

where

when	Use optional "when" clauses to indicate that the <userList> is only declined access when the named parameter <parm> (corresponding to the <parameterList> argument of the "access define" command) matches the glob-style expression given by <globExpr>. If multiple "when" clauses are used, all of them must match in order for the access rights to be affected. This is because "when" clauses are joined with an implicit AND operator.

RETURN VALUE

```
none
```

SEE ALSO

```
access allow, access deny, access filter, access reset, access verify
```

EXAMPLES

```
Let's say you want to restrict access to module data based on the
natural path of the module's member objects. Granting or denying access
at the module level is not sufficient. Access must be checked for each
of the individual module members.

In the example below:
- Developers Ian, Mahesh, Dana and Larry are the only users granted
  Checkout access to all modules.
- Test engineer Dave and doc writer Linda are declined Checkout access
  to all modules. That decline causes module member access checks to
  occur.
- Linda is explicitly granted module member checkout access to member
  objects in modules' "doc" directories.
- Dave is explicitly granted module member checkout access to member
  objects in modules' "test" directories.

access allow Checkout only users {ian mahesh dana larry} when \
      Object sync:///Modules/*
access decline Checkout users {dave linda} when \
      Object sync:///Modules/*
access allow MemberCheckout users linda when NaturalPath *doc*
access allow MemberCheckout users dave when NaturalPath *test*
```

access define

access define Command

NAME

```
access define       - Define additional actions to be access controlled
```

DESCRIPTION

```
Use the 'access define' command within to specify new actions to be
access controlled.

The pre-defined access control files included with DesignSync define
the actions for which you can control access. See the ENOVIA
Synchronicity Access Control Guide for descriptions of the
```

```
pre-defined action definitions and details on setting up and using
access controls.

You use the 'access define' command only to define additional actions
when performing advanced access rights checking.
```

SYNOPSIS

```
access define <action> [<parameterList>]
```

OPTIONS

- action Option
- parameterList Option

```
Note: For list parameters, such as <parameterList>, surround
multiple values with braces, for example, {DRCCheck ERCCheck}.
The braces are optional when you specify a single value; for example,
both {DRCCheck} and DRCCheck are valid.
```

action Option

```
action          The name of a new action to be defined. The default
                actions are defined in the default Synchronicity
                access control file by 'access define'
                statements. and have names such as Checkin, Checkout,
                and Delete. Note: Actions are case sensitive.
```

parameterList Option

```
parameterList   A list of the arguments required by any access filter
                scripts or when clauses in 'access allow' or
                'access deny' commands.
```

RETURN VALUE

```
none
```

SEE ALSO

```
access allow, access decline, access filter, access reset, access verify
```

EXAMPLES

```
See the ENOVIA Synchronicity Access Control Guide for an example of a
custom action definition.
```

access deny

access deny Command

NAME

```
access deny         - Denies access to the specified actions
```

DESCRIPTION

```
See the "access allow" command.
```

SYNOPSIS

```
access {allow | deny} <actionList> {everyone | [only] users <userList>}
[when <parm> <globExpr> [when ...]] [-because "<message_string>"]
```

access filter

access filter Command

NAME

```
access filter       - Specify criteria for allowing or denying access
```

DESCRIPTION

```
Use 'access filter' in situations where you need to allow, deny or
decline access based on criteria other than simple pattern matching of
the parameters.

To create filters where multiple notes are evaluated for ViewNote or
EditNote actions, you should use the access db_filter command to
improve performance.

See the ENOVIA Synchronicity Access Control Guide for details on
setting up and using access controls.
```

SYNOPSIS

```
access filter <actionList> [when <parm> <globExpr> [when ...]]
              <script>
```

OPTIONS

- actionList
- script
- when

Note: For list parameter <actionList>, surround multiple values with braces, for example, {Checkin Checkout}. The braces are optional when you specify a single value; for example, both {Checkout} and Checkout are valid.

actionList

actionList	The name of the actions to be controlled. The actions are defined in the default Synchronicity access control files by 'access define' statements and have names such as Checkin, Checkout, and Delete. Note: Actions are case sensitive.

script

script	A Tcl script or Tcl statements supplied directly to the 'access filter' command evaluated in order to determine if a given user should be allowed or denied access to the named action. When invoked, <script> is passed a parameter named <user> which holds the name of the user whose access is in question, in addition to the parameters listed in the access define statement that defined the action. You can also pass the parameter <action> which lets you differentiate between actions, such as Checkin or Checkout. Using the <action> parameter, you can invoke different commands depending on which action has triggered the filter statement. For example, if the action is a Checkin, you might want to check the <CommentLen> parameter is a specific length. If the action is a Checkout, you might want to allow only administrators to lock the objects. The order in which a filter is declared relative to the 'access deny' and 'access allow' commands is

important. Details are in the "Access Control Search Order" section of the topic "Setting Up Access Controls", in the ENOVIA Synchronicity Access Control Guide.

The filter script must return a value of ALLOW, DENY, DECLINE, UNKNOWN, or a message string indicating that an action is denied and explaining why. If a message string was not specified, when an action is denied, only the default "Permission denied by the AccessControl system" message is returned.

The return value of ALLOW allows the action, overriding DENY values already processed for the same operation. Likewise, the return value of DENY prevents the action, overriding ALLOW values already processed for the action. Unlike ALLOW and DENY, the return value of UNKNOWN causes the access control system to continue as if the filter had never been invoked.

The return value of DECLINE causes the appropriate Member access control to be called. The effective return value is then the result of that Member access control. Or, if there is no Member access rule, then DENY is returned.

In general, set the overall access controls you want to enforce. Then use an access filter to return ALLOW or DENY if you want to explicitly override those default rules. Otherwise, return UNKNOWN.

Note: Any value other than ALLOW, DENY, DECLINE or UNKNOWN is treated as DENY. Any uncaught exceptions are treated as DENY. For example, if you return a message string rather than DENY, the action is denied and users receive the string as an error message in addition to the default "Permission denied by the AccessControl system" message.

when

when — Use optional 'when' clauses to indicate that the user is only allowed or denied access when the named parameter <parm> (corresponding to the <parameterList> argument of the 'access define' command) matches the glob-style expression given by <globExpr>.

If multiple 'when' clauses are used, all of them must match in order for the access rights to be affected; in other words, 'when' clauses are joined with an implicit AND operator.

RETURN VALUE

```
none
```

SEE ALSO

```
access allow, access deny, access db_filter, access global,
access init, access reset, access verify
```

EXAMPLES

```
The following skeleton example uses the wildcard character passed
to access verify to ensure that notes for which users have only
partial delete access are not entirely suppressed from the GUI:

access filter DeleteNote when type ... {
  if {$id == "*"} {
   return ALLOW
  }
  ...
}

For other examples of using "access filter", see the Access Control
Guide.
```

access global

access global Command

NAME

```
access global       - Defines global variables and procs for
                      access filters
```

DESCRIPTION

```
Access filters let you allow or deny access based on criteria
beyond simple pattern matching of the parameters. You define static
variables and procs within 'access global' commands. (See 'access
init' for defining dynamic data.) To use a global variable in an
access filter, it must be declared as a global within the filter.

Unlike a normal Tcl script, the AccessControl files (both system
and custom) are sourced only once, at startup. When an 'access verify'
command is executed, only those commands given as part of access filter
scripts are evaluated. Therefore, any variables or procs defined
outside of the access commands are unknown by the Tcl interpreter. Do
not define any code or data outside either an 'access global' or
'access init' block.
```

For variables and procs to be available to the access filter scripts, they must be defined within the <script> parameter passed to either 'access global' or 'access init'. Each 'access global' block is sourced once when the access control system initializes and at every "access reset" command. The 'access global' scripts are evaluated in the order in which they appear in the custom AccessControl file.

You cannot specify a list of actions for the 'access global' command (unlike 'access init'). The 'access global' scripts are executed for all actions.

See the ENOVIA Synchronicity Access Control Guide for details on setting up and using access controls.

Important: The 'access global' code block runs inside a Tcl namespace called '::SyncAC'. Any variables that the global code block sets or procs that it defines reside in that namespace. When you use these variables or procs in other access filters, qualify them using the '::SyncAC' namespace qualifier, rather than the Tcl global qualifier '::'. Note that access filters let you qualify a variable or proc with the Tcl global namespace '::'. However, in this case, the variable or proc resides in the global namespace which is reset upon every request. Thus, these variables and procs are not visible to subsequent access filters. Note that 'access init' blocks reside in the global namespace, thus they do not need the '::SyncAC' namespace qualifier. See the EXAMPLES section for an example of the '::SyncAC' namespace qualifier.

SYNOPSIS

```
access global <script>
```

OPTIONS

- script

script

script	A Tcl script or Tcl statements that initialize the variables and procs used in the 'access filter' command for the specified actions.

RETURN VALUE

none

SEE ALSO

```
access filter, access init, access allow, access decline, access reset,
access verify
```

EXAMPLES

```
This example shows how to use 'access global' and demonstrates how
the 'access global' command differs from 'access init.' The example
sets up two lists of administrators, globalAdmins and filterAdmins.
The globalAdmins list is set up in an 'access global' block; the
filterAdmins list is set up in an 'access init' block. The AddNote
access filter demonstrates the use of the two lists. If a user is
contained in either of the lists, the filter allows permission.
Otherwise, the filter denies permission.

To use the globalAdmins list in an access filter, it must be first be
declared as a global variable in the filter. Anything declared inside
an access global block resides in the ::SyncAC namespace rather
than the global namespace (::) and needs to be referenced inside
access filters using the ::SyncAC namespace qualifier.

The filterAdmins variable does not need to be declared in the access
filter because it is set up in an 'access init' block, which is sourced
at the time the filter is run and is in the global scope of the filter.

 access global {
       set globalAdmins "norm barb mitch betty"
 }

 access init {
        set filterAdmins "mark deb sal"
 }

 access filter AddNote {
        global globalAdmins

        if {[lsearch -exact $::SyncAC::globalAdmins $user] == -1} {
                if {[lsearch -exact $filterAdmins $user] == -1} {
                        return DENY
                } else {
                        return UNKNOWN
                }
        } else {
                return UNKNOWN
        }
 }

In the example, the return value UNKNOWN defers to any other access
controls that might be set. If no other access control denies access,
the user is allowed to add a note.

For other examples of using "access global", see the Access Control
Guide.
```

access init

access init Command

NAME

```
access init          - Defines variables and procs
```

DESCRIPTION

```
You use access init to create variables for both access filters
and for use within simple access allow/access deny rules. You also
can use access init to create procs for use in access filters.

Access filters let you allow or deny access based on criteria
beyond simple pattern matching of the parameters. You define
dynamic variables and procs for access filters within 'access init'
commands. (See 'access global' for defining static data.) When
possible, you should use access global to avoid performance penalties.
Because an access init statement is sourced each time a filter
is run, operations such as viewing a note can become unacceptably
slow. The access global command, which is used inside filter scripts,
is sourced only once, when the access control system is initialized.

Unlike a normal Tcl script, the AccessControl files (both system
and custom) are sourced only once, at startup. When an 'access verify'
command is executed, only those commands given as part of access filter
scripts are evaluated; any variables or procs defined outside of the
access commands are unknown by the Tcl interpreter. Do not
define any code or data outside either an 'access init' or
'access global' block.

In order for variables and procs to be available to access filter
scripts, they must be defined within the <script> parameter passed
to 'access init' or 'access global'. The 'access init' scripts
are evaluated in the order in which they appear in the custom
AccessControl files.  By default, all 'access init' scripts are
evaluated before any 'access filter' script is evaluated.  However,
you can specify that an 'access init' script be evaluated only before
access filters verifying a particular type of action.  To do
this, include the type of action within the <actionList>
parameter of the 'access init' statement. See the example of
an <actionList> within the 'access init' example below.

Although code in 'access init' statements is executed when filters are
run, it also is available immediately for use by access allow/access
deny rules that follow the 'access init' definition.

See the ENOVIA Synchronicity Access Control Guide for details on
setting up and using access controls.
```

SYNOPSIS

```
access init [<actionList>] <script>
```

OPTIONS

- actionList
- script

```
Note: For list parameter <actionList>, surround multiple values with
braces, for example, {Checkin Checkout}. The braces are optional when
you specify a single value; for example, both {Checkout} and Checkout
are valid.
```

actionList

```
actionList      The name of the access filter actions for which the
                access init <script> will be sourced.  If no
                actionList is provided, the 'access init' script
                is evaluated before any access filter is executed
                and the script is executed for all actions.
```

script

```
script          A Tcl script or Tcl statements that initialize
                the variables and procs used in the 'access filter'
                command for the specified actions.
```

RETURN VALUE

```
none
```

SEE ALSO

```
access filter, access global, access allow, access decline,
access reset, access verify
```

EXAMPLES

```
This example shows an 'access init' statement that includes an
<actionList> argument.  The 'access init' statement is defined
for the Checkin action; thus, this 'access init' is evaluated
only before Checkin access filters.  If the <actionList> argument
(Checkin) were not included, the 'access init' would be evaluated
before any access filters are evaluated.
```

```
# Set up a variable that defines the project leader
access init Checkin {
   set projectLeader karen
}

# Only the project leader can check in
access filter Checkin {
   if {$user == $projectLeader} {
      return ALLOW
   }
   return "You must be projectleader to check in."
}
```

For other examples of using "access init", see the Access Control Guide.

access list

access list Command

NAME

```
access list          - Returns a list of defined access controls
```

DESCRIPTION

The access list command returns a list of defined access actions.

SYNOPSIS

```
access list -actions
```

OPTIONS

- -actions

-actions

```
-actions             The list of defined access action types.
```

RETURN VALUE

A space delimited list of the defined access controls appropriate to the options selected.

SEE ALSO

```
access allow, access deny, access define
```

EXAMPLES

```
access list -actions
returns "Mkmod DeleteMirror EditMirror SwitchLocker ..."
```

access reset

access reset Command

NAME

```
access reset        - Updates a SyncServer's access controls
```

DESCRIPTION

```
This server-side command causes the SyncServer to reread the
AccessControl files, which causes any changes in access controls
to take effect. All processes of a multi-process server are
affected by 'access reset', not just the process that receives the
request.

A SyncServer reads the AccessControl files upon startup, so
stopping and restarting the SyncServer also updates access
controls. Using 'access reset' avoids having to stop and restart
the SyncServer, thereby not interrupting users' access to the
SyncServer.

To update a SyncServer's access controls:

1. Modify the access control files as needed.

   See the ENOVIA Synchronicity Access Control Guide for details on
   setting up and using access controls.

2. Create a file with a .tcl extension containing the
   'access reset' command in one of the Synchronicity Tcl script
   directories:

   Site-wide:
      <SYNC_SITE_CUSTOM>/share/tcl
      (where <SYNC_SITE_CUSTOM> defaults to <SYNC_CUSTOM_DIR>/site)
   Server-specific (UNIX only):
      <SYNC_CUSTOM_DIR>/servers/<host>/<port>/share/tcl
```

```
       (where <SYNC_CUSTOM_DIR> defaults to <SYNC_DIR>/custom)

 3. Execute the script on the server using one of the following methods:

    o From your browser, specify the following URL:
      http://<host>:<port>/scripts/isynch.dll?panel=TclScript&file=<filename>

    o From a DesignSync client, specify the following command:
      rstcl -server sync://<host>:<port> -script <filename>

    where <filename> is the name of your script containing the
    'access reset' command.

 See the "server-side" and "rstcl" help topics for additional
 information on executing server-side scripts.

 Note:

 - ProjectSync provides an Access Reset option on the ProjectSync
   menu. Use this method for resetting access controls if you are a
   ProjectSync user and have privileges to use the Access Reset
   option (the Access Reset option from ProjectSync can be access
   controlled).

 - When you execute the script containing the access reset command,
   the access reset command checks the AccessControl file for
   syntactic errors before changing the server state. If the file
   contains such errors, the command aborts, leaving the server
   state (and access controls) unchanged.

 - Errors in an AccessControl file can cause your server to become
   unresponsive if the errors are not corrected quickly. To avoid this
   problem, correct access control errors immediately and reset the
   server.
```

SYNOPSIS

```
 access reset
```

OPTIONS

```
 none
```

RETURN VALUE

```
 none
```

SEE ALSO

```
stcl, rstcl, server-side, access verify
```

EXAMPLES

```
This example updates the holzt:2647 server with a site-wide
access control that stops users from unlocking files they do
not own.

1. Add the following line to <SYNC_SITE_CUSTOM>/share/AccessControl:
     access deny Unlock everyone when IsLockOwner "no"

   Note: <SYNC_SITE_CUSTOM> defaults to <SYNC_CUSTOM_DIR>/site which
   defaults to <SYNC_DIR>/custom/site.

2. In <SYNC_SITE_CUSTOM>/share/tcl, create 'reset.tcl' that
   contains the following line:
     access reset

3. Execute the script using one of the following methods:
   o From your browser, issue the following URL:
     http://holzt:2647/scripts/isynch.dll?panel=TclScript&file=reset.tcl

   o From a DesignSync client (dssc in this example), execute the
     following command:
     dss> rstcl -server sync://holzt:2647 -script reset.tcl
```

access verify

access verify Command

NAME

```
access verify       - Determines whether a user is allowed to
                      perform an action
```

DESCRIPTION

```
This server-side command checks whether the given user is allowed
to perform the named action.  Use 'access verify' to check access
controls explicitly in server-side scripts; server-side scripts
do not automatically perform the named action.

For ViewNote, EditNote, and DeleteNote filters, you can use a special
application of access verify to determine whether a user can access
any notes of that type. You can pass in an asterisk as a wildcard for
the value of a note ID. Any ViewNote, EditNote, or DeleteNote
allow/deny/filter rule treats this wildcard specially and answers
whether the specified user can access notes of that type. If the
answer is DENY, then the user does not see the note type name
```

```
in certain contexts (e.g., it disappears from QuickView because
the user does not have the right to access any notes of that type).
See the entries for access allow, access deny, and access filter for
examples of how to use the wildcard in scripts with these commands.
```

SYNOPSIS

```
access verify <action> <user> [<arg> [...]] [-why <var>]

access verify <note_action> <note_system> <note_type> *
```

ARGUMENTS

action

```
action          The name of the action to be checked. The actions are
                defined in the access control file by "access define"
                statements and have names such as Checkin, Checkout,
                and Delete.
```

args

```
args            The definition of the action in the access control
                file includes a list of parameters that give additional
                information about the action. These parameters are used
                in 'when' clauses and filter scripts in the access
                control file to determine if access will be allowed
                or denied. You must pass a value for each of these
                parameters to the access verify command to make this
                information available to the commands in the access
                control file.
```

note_action

```
<note_action>   The ViewNote, EditNote, or DeleteNote action, when
                you want to pass as asterisk as a wildcard for the
                note ID number.
```

note_system

<note_system> The name of the note system, which is always SyncNotes.

note_type

<note_type>   The name of the note type to be checked for access.

*   A wildcard character that stands for any note ID number.

user

user   The user name of the user whose access to the given
               action is being determined.

-why

-why   The definitions of access allow, deny, and filter
               statements in the access control file can include
               message strings describing why access has been denied.
               Use the -why option to retrieve this message from an
               action's access allow, deny, or filter statement and
               store it in the named variable. You can also set the
               variable to an appropriate message string directly.
               If the access statement provides no message string,
               the named variable is set to "access denied".
               If the named variable already exists and the access
               statement provides no message string, the variable's
               value remains unchanged.

See the ENOVIA Synchronicity Access Control Guide for information on
the pre-defined access control files.

RETURN VALUE

0 (Tcl FALSE) - user is not allowed access to the named action.
1 (Tcl TRUE) - user is allowed access to the named action.
2 - additional rules need to be invoked, to determine whether the user
    has access to the named action.

SEE ALSO

```
access allow, access decline, access deny, access filter, stcl, rstcl,
server-side
```

EXAMPLES

Verifying Access Rights for a User

```
This example determines whether user 'joe' has access rights to
add the version tag "Release" to top.v. Note that you do not specify a
host:port in the Object (top.v) URL (see the "server-side" help topic
for more information).

 if {[access verify Tag $SYNC_User {sync:///Projects/ASIC/top.v} \
     Release ADD VERSION]} {
   puts "$SYNC_User is allowed to tag top.v Release"
 } else {
   puts "Nice try, $SYNC_User"
 }
```

Using the Default "why" Message

```
This example shows the default message issued because the variable,
message, is not set. If a message is included in the -because or return
clause of the access allow, deny, or filter statement for the
action, the named variable is set to that message.

# Just use the default message.
# output -> "access denied"
  unset message
  if {![access verify CustomTag $SYNC_User -why message]} {
      puts $message
  }
```

Using a Custom "why" Message

```
This example shows how to use the variable, message, to issue a
descriptive message explaining why access is denied. If a message is
included in the -because or return clause of the access allow, deny,
or filter statement for the action, the named variable is set to that
message.

# Use a descriptive default message.
```

```
# output -> "Sorry, only project leader can perform CustomTag."
  set message "Sorry, only project leader can perform CustomTag."
  if {![access verify CustomTag $SYNC_User -why message]} {
      puts $message
  }
```

Using a Wildcard for note ID

```
This example passes a wildcard as the value of the note ID and
returns if access is not granted.

if {![access verify ViewNote $SYNC_User SyncNotes $noteType *]} {
    return
}
```

Authentication

hcm addlogin

hcm addlogin Command

NAME

```
hcm addlogin        - Stores a server login to enable hierarchical
                      queries (legacy)
```

DESCRIPTION

```
This command stores a login (username and password), or modifies an
existing stored login, on a server, for the purposes of conducting
cross-server recursive module queries.

The hierarchy of modules for which a query applies is defined using
hcm addhref commands. With the hcm addhref command, you can create
connections between two servers, the origin server (specified by the
-fromtarget option) and the referenced server (specified by the
-totarget option). The origin server is the server that holds the
topmost module in a design hierarchy and the server from which the
hierarchical query is initiated. You can run cross-server
hierarchical queries using ProjectSync's standard Query panel.

You need to store a login if the referenced server requires a login
different from the one you use to connect to the origin server.  For
example, your login on the origin server is "john," but your login
on the referenced server is "jdoe."

Notes:
- If you do not store a login and one is required, your query
```

```
  fails and refers you to this command.
- After you run the hcm addlogin command, you will be prompted to
  enter a password.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

This command supports the command defaults system.
```

SYNOPSIS

```
hcm addlogin -fromtarget <origin_server_url>
             -fromuser <origin_username> | -fromallusers
             -totarget <referenced_url> | -toalltargets
             -touser <referenced_username>
```

OPTIONS

- -fromallusers
- -fromtarget
- -fromuser
- -toalltargets
- -totarget
- -touser

-fromallusers

```
-fromallusers          Specifies that the stored login applies to all
                       users on the origin server.  A login stored
                       using -fromuser takes precedence over one stored
                       with -fromallusers.
```

-fromtarget

```
-fromtarget            Specifies the URL of a server on which you want
 <origin_server_url>   to store a login.

                       To specify the origin server, use the
                       following syntax:
                        sync[s]://<host>[:<port>]
                       where <host> is the machine on which the
                       SyncServer is installed and <port> is the
                       SyncServer port number.
```

-fromuser

```
 -fromuser              Specifies the username on the origin server for
  <origin_username>     which this stored login applies.
```

-toalltargets

```
 -toalltargets          Specifies that the stored login applies to all
                        servers referenced from the origin server.  A
                        login stored using -totarget takes precedence
                        over one stored with -toalltargets.
```

-totarget

```
 -totarget              Specifies that the stored login should be used
  <referenced_url>      whenever the origin server (specified by
                        -fromtarget) contacts the referenced server
                        (specified by -totarget).

                        To specify the referenced server, use the
                        following syntax:
                         sync[s]://<host>[:<port>]
                        where <host> is the machine on which the
                        SyncServer is installed and <port> is the
                        SyncServer port number.

                        Note:
                        - The value specified for
                          <referenced_server_url> should match the URL
                          returned by the hcm showhrefs command run on
                          the origin server.
```

-touser

```
 -touser                Specifies the username that HCM uses when
  <referenced_username> contacting the referenced server.

                        Note:
                        - Usernames are case sensitive.
```

RETURN VALUE

```
 This command does not return Tcl values.
```

SEE ALSO

```
 access allow, access deny, addhref, hcm rmlogin, hcm showlogins,
 command defaults
,
```

EXAMPLES

Example of Storing a User Login for a Specific Server

```
 This example stores a login 'queryuser' that users on SyncServer
 sync://chip.ABCo.com:2647 can use to query the SyncServer
 sync://alu.ABCo.com:2647.

 dss> hcm addlogin -fromtarget sync://chip.ABCo.com:2647 -fromallusers \
          -totarget sync://alu.ABCo.com:2647 -touser queryuser
```

Example of Storing a Guest Login For All Referenced Servers

```
 This example stores a login 'guest' that users on SyncServer
 sync://chip.ABCo.com:2647 can use to query all referenced servers
 (with the exception of SyncServer sync://alu.ABCo.com:2647).

 dss> hcm addlogin -fromtarget sync://chip.ABCo.com:2647 -fromallusers \
          -toalltargets -touser guest

 Note: The "Controlling Access to Servers" example in the hcm Example
 topic includes the steps for displaying and removing the stored
 logins created in the previous examples.
```

hcm rmlogin

hcm rmlogin Command

NAME

```
 hcm rmlogin          - Removes a login stored on a server
```

DESCRIPTION

```
 This command removes a login (username and password) that is stored on
 the server. This login is used for conducting cross-server recursive
 queries.

 This command is subject to access controls on the server. See the
```

```
ENOVIA Synchronicity Access Control Guide for details.

This command supports the command defaults system.
```

SYNOPSIS

```
hcm rmlogin -fromtarget <origin_server_url>
            -fromuser <origin_username> | -fromallusers
            -totarget <referenced_url> | -toalltargets
```

OPTIONS

-fromallusers

```
-fromallusers
                      Specifies that the stored login you want to
                      remove applies to all users on the origin
                      server.
```

-fromtarget

```
-fromtarget           Specifies the URL of the server from which you
 <origin_server_url>  want to remove a login.

                      To specify the origin server, use the
                      following syntax:
                       sync[s]://<host>[:<port>]
                      where <host> is the machine on which the
                      SyncServer is installed and <port> is the
                      SyncServer port number.
```

-fromuser

```
-fromuser             Specifies the username on the origin server for
 <origin_username>    which you want to remove a stored login.

-toalltargets         Specifies that the stored login you want to
                      remove applies to all servers referenced from
                      the origin server.
```

-totarget <referenced_url>	Specifies the referenced server of the stored login that you want to remove. To specify the referenced server, use the following syntax: sync[s]://<host>[:<port>] where <host> is the machine on which the SyncServer is installed and <port> is the SyncServer port number.

RETURN VALUE

This command does not return Tcl values.

SEE ALSO

access allow, access deny, hcm addlogin, hcm showlogins, command defaults
,

EXAMPLES

- Example of Removing a User Login for a Specific Server
- Example of Removing the Guest Login for a Specific Server

Example of Removing a User Login for a Specific Server

This example removes the 'queryuser' login stored on the server sync://chip.ABCo.com:2647.

```
dss> hcm rmlogin -fromtarget sync://chip.ABCo.com:2647 -fromallusers \
         -totarget sync://alu.ABCo.com:2647
```

Example of Removing the Guest Login for a Specific Server

This example removes the 'guest' login stored on the server sync://chip.ABCo.com:2647.

```
dss> hcm rmlogin -fromtarget sync://chip.ABCo.com:2647 -fromallusers \
         -toalltargets
```

hcm showlogins

hcm showlogins Command

NAME

```
hcm showlogins      - Displays the logins stored on a server
```

DESCRIPTION


```
This command displays the logins that are stored on the server. These
logins (username and password) are used for conducting cross-server
recursive queries. By default, this command returns all the login
information for the specified server to the screen in a user-friendly
format.

Note:
- The output of this command does not include passwords.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

This command supports the command defaults system.
```

Understanding the Output

```
By default, or if you run the hcm showlogins command with the '-report
normal' option, the command returns the login information to the screen
in a user-friendly format.

If you run the hcm showlogins command with the '-report script' option,
the command returns a Tcl list in the following form:

   {{fromtarget <origin_server_url>
     fromuser <origin_server_username>  totarget <reference_server_url>
     touser <reference_server_username>}
    ...
   }

   - fromtarget is the server on which the logins are stored.

   - fromuser is the user(s) to which the login applies.
     o 'ALLUSERS' indicates the login applies to all users.
     o <username> indicates the login applies to the specific user
       name.

   - totarget is the referenced server to which the access applies.
     o 'ALLTARGETS' indicates the login applies to all referenced
       servers.
     o A specific URL indicates the login applies to just that server.

   - touser is the user name to which the access applies.
```

```
If you run the hcm showlogins command with the '-report command'
option, a list of stored logins is returned in the following form:

   -fromtarget <origin_server_url>
          -fromallusers | -fromuser <origin_server_username>
          -toalltargets | -totarget <origin_server_url>
          -touser <reference_server_username>
    ...

   - fromtarget, fromallusers, fromuser, toalltargets, totarget, and
     touser are the same for both the '-report script' and '-report
     command' options. For a description of these values, see the
     description of the '-report script' output.
```

SYNOPSIS

```
hcm showlogins -fromtarget <origin_server_url>
               [ -fromuser <origin_username> | -fromallusers ]
               [ -report {brief | normal | verbose | command |
                 script}]
```

OPTIONS

- -fromallusers
- -fromtarget
- -fromuser
- -report

-fromallusers

```
-fromallusers          Displays the stored logins that apply to all
                       users on the origin server.
```

-fromtarget

```
-fromtarget            Specifies the URL of a server for which you want
  <origin_server_url>  to view logins.

                       To specify the server, use the following syntax:
                        sync[s]://<host>[:<port>]
                       where <host> is the machine on which the
                       SyncServer is installed and <port> is the
                       SyncServer port number.
```

-fromuser

-fromuser <origin_username>	Displays the stored logins for the username on the server.

-report

-report <mode>	Indicates the format in which the output appears. Valid values are: o brief - Displays the same information as 'normal'. o normal - Displays the output in a user-friendly format. This is the default behavior. o verbose - Displays the same information as 'normal'. o command - Returns a list of stored logins in a format that closely matches the original usage of the hcm addlogin and hcm rmlogin commands. With this format you can easily cut and paste to change or remove a stored login. o script - Returns a Tcl list.

RETURN VALUE

If you run the hcm showlogins command with the '-report script' option, it returns a Tcl list. If you run it with any other -report option, it does not return any Tcl values. For a description of the output, see the "Understanding the Output" section.

SEE ALSO

access allow, access deny, hcm addlogin, hcm rmlogin, command defaults
,

EXAMPLES

This example displays all the logins stored on the server sync://chip.ABCo.com:2647

```
dss> hcm showlogins -fromtarget sync://chip.ABCo.com:2647
```

password

password Command

NAME

```
password            - Stores a user's name and password
```

DESCRIPTION

```
This command allows you to save a userID and password for a DesignSync
 server or a 3DPassport Central Authentication Server, eliminating the
 need to manually authenticate.  This allows the user to run
 background jobs without requiring user input.

 The username, password, and server information is saved in the user's
 registry, the UserRegistry.reg file.  Whenever the user accesses the
 specified server from any system, the saved password is used.

 Note: The saved login information is sent to the server for all
 queries, even if authentication isn't required.  If authentication is
 not required, this command succeeds regardless of the supplied values.

 To use the command, enter the server on the command line with the
 -save option, then enter the username and password when prompted.

 When using the password command with the 3DPassport server, the login
 is persistent for the command-line and graphical clients. This
 provides the ability to use the 3DPassport single-signon
 functionality.  For information on using 3DPassport, see the
 DesignSync Data Manager Administrator's Guide: Enabling 3DPassport.

 Note: The password command does not save the password for Web
 authentication.  If you are using a web-based application, you may
 need to log in again through that interface.
```

SYNOPSIS

```
password -save <ServerURL>
```

OPTIONS

- save_option

```
-save          Specify the DesignSync URL for the server connection
 <serverURL>   appropriate for the username/password you are saving
```

```
 in the form:
 sync[s]://<host>[:<port>]
 where <host> is the hostname of the SyncServer and
<port> is the SyncServer port number.
```

RETURN VALUE

```
This command does not return any TCL values.  If the server cannot be
reached, or the account does not exist on the server, the password is
saved but not authenticated.
```

SEE ALSO

```
dssc, stclc, dss, stcl
```

EXAMPLES

```
This example shows how to save the username and password for the
specified server.

dss> password -save sync://srv2.ABCo.com:2647
Please enter account information for Synchronicity, host srv2.ABCo.com:2647.
Username: rsmith
Password: ***********
Password confirmed and saved.
```

Command Defaults

defaults Command

NAME

```
defaults            - Commands for the command defaults system
```

DESCRIPTION

```
The "defaults" commands are used to set default values for command
options, for commands run from the command line. For general
information about the command line defaults system, in a DesignSync
command shell, enter: help "command defaults"

To display a list of available "defaults" commands, in a DesignSync
```

```
command shell, enter: defaults <Tab>
```

SYNOPSIS

```
defaults <defaults_command> [<defaults_command_options>]

Usage: defaults [commands|off|on|refresh|set|show|state]
```

OPTIONS

```
Vary by command.
```

RETURN VALUE

```
Varies by command.
```

SEE ALSO

```
defaults commands, defaults off, defaults on, defaults refresh,
defaults set, defaults show, defaults state, command defaults
```

EXAMPLES

```
See specific "defaults" commands.
```

Understanding Command Defaults

command defaults Command

NAME

```
command defaults    - The command line defaults system
```

DESCRIPTION

```
Default values can be set for command options, for commands run from
the command line. Default values can be set for individual commands,
a family of commands (such as all "url" sub-commands), or all commands.
```

```
This simplifies invocations of commands from the command line, because
the user does not need to specify "-[option] <value>" for the saved
default value.

To display a list of available "defaults" commands, in a DesignSync
command shell, enter: defaults <Tab>

Run "defaults commands" for the list of commands that support the
command line defaults system.

Use "defaults set" to set default values. Specifying an option on the
command line overrides a saved default value for the specified option.
A saved default value takes precedence over default behavior specified
via SyncAdmin.

To set default values for a project team, or for all users at a site,
use the "sregistry scope" command.

Any command that supports the command line defaults system has a
"-nodefaults" override option. If "-nodefaults" is  specified, then any
saved default values will be ignored. The intent of the "nodefaults"
option is for script writers to ensure that functions are called as the
writer intends the functions to be called.

See the "defaults set" command documentation for how to set a default
fetch state for the command line. Saving a default value for the
"-exclude" option pertains only to the "-exclude" command line option.
It does not affect the Exclude Lists described in the SyncAdmin help.

DesignSync graphical applications are not affected by the command line
defaults system. DesignSync DFII and the DesignSync GUI do not use
saved command line default values. However, the command bar in the
DesignSync GUI does use the command line defaults system.
```

SEE ALSO

```
defaults commands, defaults off, defaults on, defaults refresh,
defaults set, defaults show, defaults state, sregistry scope
```

EXAMPLES

Example of Setting the Default

```
To set "-report status" as the default report mode for "ls":
stcl> defaults set -- ls -report status
```

Example of Showing the Saved Defaults

```
To see what default options have been saved for the "ls" command:
stcl> defaults show ls
{ls {-report status}}
```

Example of Overriding the Set Defaults for the Whole Command

```
To ignore the saved default for "ls", and use the built-in default
report mode ("-report normal"):
stcl> ls -nodefaults
```

Example of Overriding a Specific Option

```
Specifying an option value at the command line overrides the option's
saved default value. For example:
stcl> ls -report brief
```

defaults

defaults Command

NAME

```
defaults            - Commands for the command defaults system
```

DESCRIPTION

```
The "defaults" commands are used to set default values for command
options, for commands run from the command line. For general
information about the command line defaults system, in a DesignSync
command shell, enter: help "command defaults"

To display a list of available "defaults" commands, in a DesignSync
command shell, enter: defaults <Tab>
```

SYNOPSIS

```
defaults <defaults_command> [<defaults_command_options>]

Usage: defaults [commands|off|on|refresh|set|show|state]
```

OPTIONS

```
Vary by command.
```

RETURN VALUE

```
Varies by command.
```

SEE ALSO

```
defaults commands, defaults off, defaults on, defaults refresh,
defaults set, defaults show, defaults state, command defaults
```

EXAMPLES

```
See specific "defaults" commands.
```

defaults commands

defaults commands Command

NAME

```
defaults commands   - Lists the commands that support the defaults system
```

DESCRIPTION

- Note for Legacy Module Mode (Legacy-based)

```
Lists the commands that support the defaults system. Those commands can
have default values set for them, by using the "defaults set" command.
```

Note for Legacy Module Mode (Legacy-based)

```
If you have enabled the legacy command support, the hcm prefix
does not display in front of the module commands.  The legacy command
set is not supported by the command defaults system.
```

SYNOPSIS

```
defaults commands
```

RETURN VALUE

```
A list of the commands that accept default values. The order of the
entries in the list is non-deterministic. For a family of commands, its
sub-commands are listed.

For example:
    {replicate {addroot data showroots showdata rmroot rmdata disable
    reset enable setoptions scrub masrename}}
```

SEE ALSO

```
defaults on, defaults refresh, defaults set, defaults show,
defaults state, command defaults
```

EXAMPLES

```
To list all commands that support the command defaults system:

stcl> defaults commands
 vhistory rmlogin showmcache {view {check get list put remove}} tag
 mkmod {replicate {addroot data showroots showdata rmroot rmdata
 disable reset enable setoptions scrub masrename}} mvfile unfreezemod
 showstatus contents mkbranch compare hcm lock {mcache {scan touch
 scrub show}} annotate rmfolder remove addbackref upgrade add
 reconnectmod switchlocker mvmod showmods migratetag edithrefs rmmod
 exportmod ci showlogins addhref cancel setview freezemod co populate
 rmfile mvfolder showhrefs unremove importmod purge showlocks
 rmversion addlogin {sitr {env integrate lookup mkbranch mkmod
 populate release select status submit update}} rmvault mvmember
 {swap {replace restore show}} retire unlock rmhref whereused version
 ls

stcl>
```

defaults off

defaults off Command

NAME

```
defaults off        - Disables the command defaults system
```

DESCRIPTION

Disables the command default system. If you do not specify an <expr>, the command default system is disabled until a subsequent "defaults on" command is run within the current client session. When a DesignSync client is started, the command defaults system is enabled.

If you do specify an <expr>, the command defaults system is disabled for the duration of the execution of the supplied Tcl expression.

The "defaults off" and "defaults on" commands do not "nest". The caller of those commands must ensure that the command defaults system is in the correct state at any time.

Use the "defaults state" command to show whether the command defaults system is current enabled or disabled.

To disable the command defaults system when running an individual command, specify the "-nodefaults" option to the command. Use "defaults off" if you are calling a user-defined procedure or alias that itself calls one or more DesignSync commands.

SYNOPSIS

```
defaults off [<expr>]
```

RETURN VALUE

If you do not specify an <expr>, the state value "off" is returned. If you do specify an <expr>, the result of that <expr> is returned. If the <expr> throws an error, "defaults off" will throw an error.

When an <expr> is specified, the state of the command defaults system is always returned to what it was originally, even if the <expr> threw an error.

SEE ALSO

defaults commands, defaults on, defaults refresh, defaults set, defaults show, defaults state, command defaults

EXAMPLES

This example shows what happens when you set a default on a command, run the command. The example sets the default "-report verbose" for the ls command and then shows running the command with the default

```
mode disabled.

stcl> defaults set -- ls -report verbose

The "defaults show" command confirms the default setting for "ls":

stcl> defaults show ls
{ls {-report verbose}}
stcl>

"ls" of an object, without specifying a "-report" option, uses the saved
default mode of "-report verbose" (instead of the out-of-the-box default
mode of "-report normal"):

stcl> ls samp.asm
Object Type  Time Stamp             Status  Version  Locked By  Name
-----------  ----------             ------  -------  ---------  ----
File         05/25/1997 22:04   Up-to-date  1.1                samp.asm
Original Log: --> Created by tbarbg10 @05/26/2006 09:40:10
               --> Old DAC demo files
Version Tags: Latest
 Branch Tags: Trunk
stcl>

To ignore saved default values while a command is run, preface the
command invocation with "defaults off". For example, if the command
defaults system is disabled while an "ls" is run, that "ls" will use
the out-of-the-box default mode of "-report normal":

stcl> defaults off ls samp.asm
Time Stamp             WS Status  Version   Type        Name
----------             ---------  -------   ----        ----
05/25/1997 22:04                  1.1       Copy        samp.asm
stcl>
```

defaults on

defaults on Command

NAME

```
defaults on         - Enables the command defaults system
```

DESCRIPTION

```
Re-enables the command default system. The command defaults system is
enabled by default. The command defaults system remains enabled within
a client session until the "defaults off" command is run.

The "defaults off" and "defaults on" commands do not "nest". The caller
of those commands must ensure that the command defaults system is in
the correct state at any time.
```

SYNOPSIS

```
defaults on
```

RETURN VALUE

```
The state value "on".
```

SEE ALSO

```
defaults commands, defaults off, defaults refresh, defaults set,
defaults show, defaults state, command defaults
```

EXAMPLES

```
The example shows enabling and disabling the command defaults system,
and using the "default state" command to show whether defaults is
enabled.

stcl> defaults off
off
stcl> defaults state
off
stcl> defaults on
on
stcl> defaults state
on
stcl>
```

defaults refresh

defaults refresh Command

NAME

```
defaults refresh    - Refreshes the command defaults system
```

DESCRIPTION

```
Refreshes the command defaults system, by re-reading the registry files
sourced by the DesignSync client on startup. Default values set via the
```

command defaults system are used by the DesignSync client from which the default values were set. If you saved default values in concurrent DesignSync client sessions, run the "defaults refresh" command to read all saved default values.

Similarly, if default values were saved by a project lead or site administrator, run "defaults refresh" to read default values from all client registry files. See the DesignSync Data Manager User's Guide topic "Registry Files" for further information. New DesignSync client sessions read all saved default values (from registry files) on startup.

SYNOPSIS

```
defaults refresh
```

RETURN VALUE

Not defined.

SEE ALSO

defaults commands, defaults off, defaults on, defaults set, defaults show, defaults state, command defaults, sregistry scope

EXAMPLES

This example shows what happens when a default on a command in a different client than the one you're using, and you want to refresh your client to read in the new default value.

```
stcl> defaults set -- ls -report verbose
stcl>
```

"defaults show" shows that the only saved default value for "ls" is the report mode:

```
stcl> defaults show ls
{ls {-report verbose}}
stcl>
```

In another stclc session, you save another default value for "ls":

```
stcl> defaults set -- ls -report verbose -recursive
stcl>
```

Back in the stclc session from which you set the "-report verbose" default value, "-report verbose" is still the only default value saved

```
for "ls":

stcl> defaults show ls
{ls {-report verbose}}
stcl>

Still in the stclc session from which you set the "-report verbose"
default value, you run "defaults refresh", to re-read the client
registry files:

stcl> defaults refresh
stcl>

Now, that initial stclc recognizes the default values that were saved
in the other stclc session:

stcl> defaults show ls
{ls {-recursive -report verbose}}
stcl>
```

defaults set

defaults set Command

NAME

```
defaults set        - Defines the default values for a command
```

DESCRIPTION

- Note for Module Commands (Module-based)

```
Defines the default values for a command or sub-command. For a default
value to apply to all commands that support the specified option, specify
a <command> value of "*". For a default value to apply to a family of
commands, specify the parent command as the <command> value. For example,
a <command> value of "access" applies the specified default values to all
"access" sub-commands that support the specified option.

Note: The values set as the new command default REPLACE the
previously set command defaults. All previously set defaults
specified for the same command level are removed.  For example, if
you have changed the report mode for the replicate command set, it
will not remove the defaults set on the specific replicate commands.
For more clarification, see the examples section.

"defaults set" saves defaults for the user. To set default values for a
project team, or for all users at a site, use the "sregistry scope"
command. See the "sregistry scope" command documentation for details.

To set a command line default value for the fetch state, specify the
```

```
exact option name used by a command. To specify local copies as the
default fetch state value, set the "-keep" option for the "cancel" and
"ci" commands. And set the "-get" option for the "co" and "populate"
commands.

To remove the saved default values for a command, specify empty double
quotes ("") as the option value. See the Examples section for syntax.
```

Note for Module Commands (Module-based)

```
Note: You cannot set a global default value on module (hcm)
commands.  You must specify the command defaults individually;
optimally by using the command name with no hcm prefix.
```

SYNOPSIS

```
defaults set [-temporary | -nooverrule] -- <command> <option>
             [<option> ...]
```

OPTIONS

- -nooverrule
- -temporary
- --

-nooverrule

```
-nooverrule    The saved default value cannot be overridden. This option
               is intended for project leaders or site administrators.

               When using the "sregistry scope" command to set a default
               value for a project, specifying "-nooverrule" prevents
               a user's saved default value from overriding the project's
               default value.

               Similarly, when using the "sregistry scope" command to
               set a site-wide default value, specifying "-nooverrule"
               prevents project and user saved default values from
               overriding the site's default value.

               Command options specified by the user will be used, taking
               precedence over a "-nooverrule" setting.
```

-temporary

```
-temporary     The saved default value applies only to the DesignSync
```

```
                client session from which the "defaults set" command was
                run.
```

--

```
 --             Indicates that the command should stop looking for
                command options. Use this option when an argument to the
                command begins with a hyphen (-).
```

RETURN VALUE

```
 Not defined.
```

SEE ALSO

```
 defaults commands, defaults off, defaults on, defaults refresh,
 defaults show, defaults state, command defaults, sregistry scope
```

EXAMPLES

Example of Setting the Default Options for a Specific Command

```
 This example sets the following options for the ls command:
  * Sets the -report mode to "status"
  * Sets the -[no]path option to path

 stcl> defaults set -- ls -report status -path

 To see the saved defaults for the "ls" command
 stcl> defaults show ls
 {ls {-report status -path}}
```

Example of Resolving Default Conflicts

```
 This example shows how DesignSync resolves set default conflicts.
 Locally, you have set the default defined in Example 1.  Your project
 team leader uses the "sregistry scope" command to save "-fullpath" as
 a default option to "ls".
```

```
Adding the "-source" option to "defaults show" shows the two different
(mutually exclusive) options that were saved at the user level ("-path")
and at the project level ("-fullpath"):

stcl> defaults show -source ls
{ls temporary {} project -fullpath project_nooverrule {}
user {-report status -path} user_nooverrule {} site {}
site_nooverrule {} enterprise {} enterprise_nooverrule {}}

The default value saved by the user ("-path") takes precedence, because
the project leader did not specify "-nooverrule". (The project_nooverrule
value in the "defaults show" output above is empty.) That is why
"defaults show" (without "-source") still shows the default "-path"
value that you saved:

stcl> defaults show ls
{ls {-report status -path}}
```

Example of Clearing the Defaults

```
This example shows how to remove your saved default values for the
"ls" command:
stcl> defaults set ls ""

Using the settings from example 2, the default "-fullpath" value is
still valid because it was set at the project level:

stcl> defaults show ls
{ls -fullpath}
stcl> defaults show -source ls
{ls temporary {} project -fullpath project_nooverrule {} user {}
user_nooverrule {} site {} site_nooverrule {} enterprise {}
enterprise_nooverrule {}}
```

Example of Setting Defaults for All Commands

```
This example shows setting the "-recursive" option as the default
behavior for all commands that support the command defaults system
(and have a "-recursive" option):

Note: This does not remove any previously set defaults on specific
commands or command sets.  If you look at the ls command in the
"defaults show" results displayed in this example, you will see that
the command defaults set in the Example 1, for ls, are still set.

stcl> defaults set -- * -recursive

To see which commands now have "-recursive" saved as their default
behavior:

stcl> defaults show
```

```
{vhistory -recursive} {rmlogin {}} {showmcache {}} {view {}} {{view
check} {}} {{view get} {}} {{view list} {}} {{view put} {}} {{view
remove} {}} {tag {-recursive -report normal}} {mkmod {}} {replicate
{}} {{replicate addroot} {}} {{replicate data} {}} {{replicate
showroots} {}} {{replicate showdata} {}} {{replicate rmroot} {}}
{{replicate rmdata} {}} {{replicate disable} {}} {{replicate reset}
{}} {{replicate enable} {}} {{replicate setoptions} {}} {{replicate
scrub} {}} {{replicate masrename} {}} {mvfile {}} {showstatus
-recursive} {contents -recursive} {mkbranch -recursive} {compare
-recursive} {hcm -recursive} {lock {}} {mcache {}} {{mcache scan} {}}
{{mcache touch} {}} {{mcache scrub} {}} {{mcache show} {}} {annotate
{}} {rmfolder -recursive} {remove -recursive} {addbackref -recursive}
{upgrade {}} {add {-recursive -report normal}} {switchlocker {}}
{showmods {}} {migratetag {}} {rmmod -recursive} {ci {-recursive
-report normal}} {showlogins {}} {addhref {}} {cancel -recursive}
{setview -recursive} {co -recursive} {populate {-recursive -report
normal}} {mvfolder {}} {rmfile {}} {showhrefs -recursive} {unremove
{}} {purge -recursive} {rmversion {}} {showlocks -recursive}
{addlogin {}} {sitr {}} {{sitr env} {}} {{sitr integrate} {}} {{sitr
lookup} {}} {{sitr mkbranch} {}} {{sitr mkmod} {}} {{sitr populate}
{}} {{sitr release} {}} {{sitr select} {}} {{sitr status} {}} {{sitr
submit} {}} {{sitr update} {}} {mvmember {}} {swap {}} {{swap
replace} {}} {{swap restore} {}} {{swap show} {}} {retire -recursive}
{rmvault {}} {unlock -recursive} {rmhref {}} {whereused -recursive}
{version {}} {ls {-recursive -report status -path}} {* -recursive}

If you want an option, as in this example, -recursive, to be the
default for the majority of command that support it, but have
specific commands for which you would prefer a different mode, you
can set the different mode as a specific default on the command.

Note: The show output is truncated for clarify.

stcl> defaults set -- populate -norecursive

stcl> defaults show

{vhistory -recursive} {rmlogin {}} {showmcache {}} {view {}}
...
{setview -recursive} {co -recursive} {populate -norecursive}
...
{ls {-recursive -report status -path}} {* -recursive}
```

defaults show

defaults show Command

NAME

```
defaults show       - Shows the current default values for a command
```

DESCRIPTION

- Note for Module Commands (Module-based)

Shows the current default values for a command or sub-command. If no <command> is specified, then the current default values for all commands and sub-commands are reported.

A <command> value of "*" will show the current default values saved against "*". Those default values will be applied to all commands that take those options.

If a sub-command is specified as the <command> value, such as "replicate disable", default values for the specific sub-command ("replicate disable" in this case), its parent command ("replicate" in this example) and any global default values will be shown. Similarly, the results for a command combine the default values for the specific command, with any global default values.

If a sub-command is specified as the <command> value and the -source option is selected, the default command returns ONLY the default values for the specific command. The reply does not include any global defaults or any parent command defaults.

Note for Module Commands (Module-based)

Note: You cannot set a global default value on module (hcm) commands. You must specify the command defaults individually; optimally by using the command name with no hcm prefix.

SYNOPSIS

```
defaults show [-source] [<command>]
```

OPTIONS

- -source

-source

-source When specified, the output indicates where the default value is saved. By default, command default values are stored for the user who ran the "defaults set" command. Default values can also be stored for a project, or site-wide. See the "sregistry scope" command for details.

When you specify both a command and the -source option, the command output displays ONLY the command default values associated with that command. Any

values set on a parent command or globally do not display.

RETURN VALUE

A list, where each entry in the list is the current setting of defaults for a command or subcommand. The order of the entries in the list is non-deterministic.

SEE ALSO

defaults commands, defaults off, defaults on, defaults refresh, defaults set, defaults state, command defaults, sregistry scope

EXAMPLES

An example in the "defaults set" command documentation sets "-recursive" as the default behavior, for every command that supports the command defaults system (and has a "-recursive" option).

After having saved the global "-recursive" default, "defaults show" shows the default "-recursive" value for the "tag" command:

```
stcl> defaults show tag
{tag -recursive}
```

Next, let's save "-modified" as default behavior for the "tag" command:

```
stcl> defaults set -- tag -modified
```

Now, "defaults show" shows both the global "-recursive" option, and the "-modified" default value that was explicitly saved for the "tag" command.

```
stcl> defaults show tag
{tag {-recursive -modified}}
```

The above output from "defaults show" shows the combined defaults for the "tag" command. This represents exactly what options will be applied when the "tag" command is run.

Adding the "-source" option to the "defaults show" command shows where the default values for the "tag" command were saved.

```
stcl> defaults show -source tag
{tag temporary {} project {} project_nooverrule {} user -modified
user_nooverrule {} site {} site_nooverrule {} enterprise {}
enterprise_nooverrule {}}
```

The above output shows that the "-modified" option to "tag" was saved at

```
the user level. The "-recursive" option is not shown, because that option
was saved globally.

For global default values, specify "*" as the command:

stcl> defaults show -source *
{* temporary {} project {} project_nooverrule {} user -recursive
user_nooverrule {} site {} site_nooverrule {} enterprise {}
enterprise_nooverrule {}}

The above output shows that the "-recursive" option was set globally, at
the user level.
```

defaults state

defaults state Command

NAME

```
defaults state      - Returns the state of the command defaults system
```

DESCRIPTION

```
Returns whether the command defaults system is currently enabled ("on")
or disabled ("off").
```

SYNOPSIS

```
defaults state
```

RETURN VALUE

```
The state value "on" or the state value "off".
```

SEE ALSO

```
defaults commands, defaults off, defaults on, defaults refresh,
defaults set, defaults show, command defaults
```

EXAMPLES

```
This example shows the default state with the command defaults system
```

```
enabled and disabled.

stcl> defaults state
on
stcl> defaults off
off
stcl> defaults state
off
stcl>
```

Custom Type System

Custom Type Packages

ctp

ctp Commands

NAME

```
ctp                 - Commands to list and verify Custom Type Packages
```

DESCRIPTION

```
The 'ctp' commands help you to manage the installed Custom Type
Packages (CTPs).  You can list the CTPs using 'ctp list' and
debug your CTPs using the 'ctp verify' command.

A CTP is a Tcl file containing procedures that recognize and traverse
your custom data hierarchy, grouping the data into collections. You
install the CTP in one of the following Synchronicity custom hierarchy
directories:

o Project-level CTP:
  <SYNC_PROJECT_CFGDIR>/ctp/<ctp_file>.ctp

o Site-level CTP:
  <SYNC_SITE_CUSTOM>/share/client/ctp/<ctp_file>.ctp

When next you invoke a DesignSync client, the DesignSync Custom Type
System registers the CTP so that each revision control operation can
now recognize and manage the collection types defined in your CTP.

To develop a CTP for your custom data, see the DesignSync Custom Type
System Programmer's Guide.
```

SYNOPSIS

```
ctp <ctp_command> [<ctp_command_options>]
```

```
Usage: ctp [list|verify]
```

OPTIONS

```
Vary by command.
```

RETURN VALUE

```
Varies by command.
```

SEE ALSO

```
ctp list, ctp verify, localversion, localversion delete,
localversion list, localversion restore, localversion save
```

EXAMPLES

```
See specific 'ctp' commands.
```

ctp list

ctp list Command

NAME

```
ctp list            - Lists installed Custom Type Packages
```

DESCRIPTION

```
This command lists the names of all currently installed Custom Type
Packages (CTPs). You run the 'ctp list' command from any directory,
with no arguments.

A CTP is a Tcl file containing procedures that recognize and
traverse your custom data hierarchy, grouping the data into
collections. You install the CTP in one of the following
Synchronicity custom hierarchy directories:

o Project-level CTP:
  <SYNC_PROJECT_CFGDIR>/ctp/<ctp_file>.ctp

o Site-level CTP:
  <SYNC_SITE_CUSTOM>/share/client/ctp/<ctp_file>.ctp
```

When next you invoke a DesignSync client, the DesignSync Custom Type System registers the CTP so that each revision control operation can now recognize and manage the collection types defined in your CTP.

To develop a CTP for your custom data, see the DesignSync Custom Type System Programmer's Guide.

SYNOPSIS

```
ctp list
```

ARGUMENTS

None.

OPTIONS

None.

RETURN VALUE

In dss/dssc mode, you cannot operate on return values, so the return value is irrelevant.

In stcl/stclc mode, the command returns a list of the installed CTPs and an empty list if there are no installed CTPs.

SEE ALSO

ctp, ctp verify, localversion, localversion delete, localversion list, localversion save, localversion restore

EXAMPLES

The following example lists all of the Custom Type Packages (CTPs) currently installed.

```
stcl> ctp list
collectionCTP localCTP kmlocalCTP dsmwCTP
```

ctp verify

ctp verify Command

NAME

```
ctp verify          - Validates installed Custom Type Package files
```

DESCRIPTION

```
This command verifies all installed Custom Type Packages (CTPs).
A CTP is a Tcl file containing procedures that recognize and traverse
your custom data hierarchy, grouping the data into collections. You
install the CTP in one of the following Synchronicity custom hierarchy
directories:

o Project-level CTP:
  <SYNC_PROJECT_CFGDIR>/ctp/<ctp_file>.ctp

o Site-level CTP:
  <SYNC_SITE_CUSTOM>/share/client/ctp/<ctp_file>.ctp

When next you invoke a DesignSync client, the DesignSync Custom Type
System registers the CTP so that each revision control operation can
now recognize and manage the collection types defined in your CTP.

To develop a CTP for your custom data, see the DesignSync Custom
Type System Programmer's Guide.

The 'ctp verify' command tests for inconsistencies in the behavior
of related procedures in the CTPs. There are several places where a
CTP is required to return the same information in multiple places,
and these must be consistent for the CTP to work correctly. For
example, if mapViews assigns an objtype to a particular object, but
updateObject does not, then the CTP will not behave correctly. 'ctp
verify' flags this type of inconsistency.  Among the ways a CTP can
be internally inconsistent are:

o The mapViews and determineFolderType procedures return different
  values for a given folder.

o The mapViews procedure identifies an object as a member, but it is
  not returned by any collection's members procedure.

o The mapViews procedure fails to identify an object as a member when
  it is returned by a collection's members procedure.

o A collection member has an owner property identifying a collection,
  but that collection does not identify it as a member.

o More than one collection identifies a file as a member.

The 'ctp verify' command validates all of the installed CPTs against
the data in the specified path. The command lists:
```

```
o The installed and active CTPs

o The folder and subfolders being validated

o A status of the collection members in the folder and its subfolder

  The command lists the objects that are not members of any of the
  installed CTPs.  It also lists the collections that have no members.
  These occurrences might flag an error in a CTP.

It is important that you use the 'ctp verify' command to validate
your CTPs before making them available for use with production
design data. The Custom Type System checks for exceptions thrown
by particular CTP procedures, but it does not check for inconsistent
CTPs during revision control operations. These checks would
greatly diminish the efficiency of DesignSync's data traversal.

In order to fully validate your CTPs, it is important that you
develop test data that exercises all aspects of the CTPs.  See
the DesignSync Custom Type System Programmer's Guide to help
you design your test data to ensure that 'ctp verify' detects
specific error conditions.

In addition to applying 'ctp verify' to your CTPs, use the
'ls -report OX' command to list objects and their collection
owners.  Use the 'url members' command to list a collection's
members.  These commands will help ensure that your CTP
manages your data as intended.
```

SYNOPSIS

```
ctp verify [<path>]
```

ARGUMENT

- path

path

```
path          The path to the directory/folder containing the
              data used to validate the installed CTPs. You can
              specify an absolute or relative path. If no path is
              specified, 'ctp verify' validates the installed CTPs
              against the current directory.
```

OPTIONS

```
None.
```

RETURN VALUE

```
In dss/dssc mode, you cannot operate on return values, so the
return value is irrelevant.

In stcl/stclc mode, the command returns the number of errors
found.
```

SEE ALSO

```
ctp, ctp list, localversion, localversion delete,
localversion list, localversion save, localversion restore
```

EXAMPLES

Example of a Clean ctp verify

```
The following example shows verification of a clean CTP;
the 'ctp verify' command returns an error count of 0.

stcl> ctp verify
============== Verifying CTPs  ======================
The following CTPs are installed: collectionCTP dsmwCTP
Verifying at root /home/karen/sf242data/jul16/coltest

---- Verifying /home/karen/sf242data/jul16/coltest

---- Verify collection member information
0
```

Example of a cpt verify Showing Errors

```
This next example verifies a single CTP, collection.ctp:

stcl> ctp list
collectionCTP
stcl> ctp verify
============== Verifying CTPs  ======================
The following CTP's are installed: collectionCTP
Verifying at root /home/karen/sf242data/sep2/ctp_tests/coltest

---- Verifying /home/karen/sf242data/sep2/ctp_tests/coltest
ctp collectionCTP: For object 'g.sgc.tst', the property 'objtype' is
not the same:
```

```
   'Test orange' is set in the updateObject proc and
   'Test collection' is set in the mapViews proc.
ctp collectionCTP: For object 'f.sgc.tst', the property 'objtype' is
not the same:
   'Test orange' is set in the updateObject proc and
   'Test collection' is set in the mapViews proc.
ctp collectionCTP: For object 'a.sgc.tst', the property 'objtype' is
not the same:
   'Test orange' is set in the updateObject proc and
   'Test collection' is set in the mapViews proc.
ctp collectionCTP: For object 'd.sgc.tst', the property 'objtype' is
not the same:
   'Test orange' is set in the updateObject proc and
   'Test collection' is set in the mapViews proc.

---- Verify collection member information
4
```

Managing Local Versions of Collections

localversion

localversion Commands

NAME

```
localversion        - Commands to manage local versions of collections
```

DESCRIPTION

```
Some design tools implement their own basic version management
by making local copies of design objects. A local copy of a design
object is referred to as a 'local version', to distinguish it
from the DesignSync version, which is created in the DesignSync
vault upon checkin (a vault version).

Where DesignSync manages such data either through a predefined
recognition package or through a developer's
Custom Type Package (CTP), DesignSync incorporates the local version
number into a tag name it applies upon checkin of the object.
For example, if an object's local version is 6, DesignSync applies
the tag <collection_type>_6 upon checkin.

Depending upon the options you choose, the DesignSync 'co' and
'populate' commands can remove local versions of an object,
replacing them with the requested version from the DesignSync
vault. You can save the local versions using the -savelocal
option to the 'co' or 'populate' commands.  You can also save the
local versions using the 'localversion save' command and later
retrieve them using the 'localversion restore' command. The
'localversion list' command lets you view the saved local versions.
```

```
You can change the local version default behavior so that DesignSync
automatically saves all local versions before fetching. To change
this setting, a Synchronicity administrator can use the Local
Versions field on the Command Defaults pane of the SyncAdmin tool.
For information, see SyncAdmin help.
```

SYNOPSIS

```
localversion <localversion_command> [<localversion_command_options>]

Usage: localversion [delete|list|restore|save]
```

OPTIONS

```
Vary by command.
```

RETURN VALUE

```
Varies by command.
```

SEE ALSO

```
localversion delete, localversion list, localversion restore,
localversion save, ctp,
```

EXAMPLES

```
See specific 'localversion' commands.
```

localversion delete

localversion delete Command

NAME

```
localversion delete - Deletes a saved version of the given collection
```

DESCRIPTION

```
Use the 'localversion delete' command to delete a local version that
was previously saved for the specified collection. Use the
'localversion list' command to list the local versions to be deleted.
```

```
Some design tools implement their own basic version management
by making local copies of design objects. A local copy of a design
object is referred to as a 'local version', to distinguish it
from the DesignSync version, which is created in the DesignSync
vault upon checkin (a vault version).

Where DesignSync manages such data either through a predefined
recognition package or through a developer's Custom Type Package
(CTP), DesignSync incorporates the local version number into a tag
name it applies upon checkin of the object. For example, if an
object's local version is 6, DesignSync applies the tag
<collection_type>_6 upon check-in.

Note: This command only affects objects of a collection defined by
the Custom Type Package (CTP). This command does not affect objects
that are not part of a collection or collections that do not have
local versions.
```

SYNOPSIS

```
localversion delete <sync collection>|<sgc collection> <locint>
```

OPTIONS

- Synchronicity Collection
- Custom Generic Collection
- Local Version Integer

Synchronicity Collection

```
sync collection   - Specify a Synchronicity predefined collection
                    in the format <object>.sync.<collectiontype>
                    where

                    <object> is the base name of the object, for
                    example, a cell or view name.

                    sync indicates a Synchronicity predefined
                    collection.

                    <collectiontype> is the collection name.
                    Predefined collection names include cds
                    and mw.

                    Examples of Synchronicity predefined collection
                    objects include NAND.sync.mw and cell1.sync.mw.

                    You can specify the object as a local URL
                    (starts with the file:/// protocol) or as an
                    absolute or relative path.
```

Custom Generic Collection

```
sgc collection    - Specify a custom generic collection (defined
                    in a Custom Type Package) using the format
                    <object>.sgc.<collectiontype> where

                    <object> is the base name of the object, for
                    example, a cell or view name.

                    sgc indicates a custom collection defined
                    in a Custom Type Package.

                    <collectiontype> is the collection name
                    defined in a Custom Type Package (CTP).
                    See the DesignSync Custom Type System
                    Programmer's Guide for more information.
                    An example of a custom generic collection
                    object is symbol.sgc.mytool.

                    You can specify the object as a local URL
                    (starts with the file:/// protocol) or as an
                    absolute or relative path.
```

Local Version Integer

```
locint            - Specify the integer assigned to the saved
                    local version of the collection.  If you
                    do not know the integer, use the
                    'localversion list' command to view the
                    saved local version numbers.
```

RETURN VALUE

In dss/dssc mode, you cannot operate on return values, so the return value is irrelevant.

In stcl/stclc mode, the command returns the integer corresponding to the deleted local version. If a collection has no saved local versions or if the value specified for the saved local version number is not an integer, the command throws an error.

SEE ALSO

localversion, localversion list, localversion save, localversion restore

EXAMPLES

```
The following example deletes the saved local version of the
local.sgc.loc collection.  The example first lists the saved local
versions of the local.sgc.loc collection.

stcl> localversion list .
local.sgc.loc {1 2 3}
stcl> localversion delete local.sgc.loc 1
1
stcl> localversion list .
local.sgc.loc {2 3}
```

localversion list

localversion list Command

NAME

```
localversion list   - Lists saved versions of collection objects
```

DESCRIPTION

```
Use the 'localversion list' command to list the local versions that
were previously saved for the specified collection. You can instead
specify a directory name to list all of the collections with
previously saved local versions in that directory. In this case,
each collection is listed, followed by a list of its local version
numbers.  If a collection has no saved local versions or if
a directory contains no collections with saved local versions,
the 'localversion list' command returns an empty list.

Some design tools implement their own basic version management
by making local copies of design objects. A local copy of a design
object is referred to as a 'local version', to distinguish it
from the DesignSync version, which is created in the DesignSync
vault upon checkin (a vault version).

Where DesignSync manages such data either through a predefined
recognition packageor through a developer's
Custom Type Package (CTP), DesignSync incorporates the local version
number into a tag name it applies upon checkin of the object.
For example, if an object's local version is 6, DesignSync applies
the tag <collection_type>_6 upon checkin.

Note:
 This command only affects objects of a collection defined by the
 Custom Type Package (CTP). This command does not affect objects that
 are not part of a collection or collections that do not have local
 versions.
```

SYNOPSIS

```
localversion list <sync collection>|<sgc collection>|<directory>
```

OPTIONS

Synchronicity Predefined Collection

```
sync collection    - Specify a Synchronicity predefined collection
                     in the format <object>.sync.<collectiontype>
                     where

                     <object> is the base name of the object, for
                     example, a cell or view name.

                     sync indicates a Synchronicity predefined
                     collection.

                     <collectiontype> is the collection name.
                     Predefined collection names include cds
                     and mw.

                     Examples of Synchronicity predefined collection
                     objects include NAND.sync.mw and cell1.sync.mw.

                     You can specify the object as a local URL
                     (starts with the file:/// protocol) or as an
                     absolute or relative path.
```

Custom Generic Collection

```
sgc collection     - Specify a custom generic collection (defined
                     in a Custom Type Package) using the format
                     <object>.sgc.<collectiontype> where

                     <object> is the base name of the object, for
                     example, a cell or view name.

                     sgc indicates a custom collection defined
                     in a Custom Type Package.

                     <collectiontype> is the collection name
                     defined in a Custom Type Package (CTP).
                     See the DesignSync Custom Type System
                     Programmer's Guide for more information.
```

```
                    An example of a custom generic collection
                    object is symbol.sgc.mytool.

                    You can specify the object as a local URL
                    (starts with the file:/// protocol) or as an
                    absolute or relative path.
```

Collection Directory

```
directory         - Specify a directory containing a collection.
                    You can specify the directory as a local URL
                    (starts with the file:/// protocol) or as an
                    absolute or relative path.
```

RETURN VALUE

```
In dss/dssc mode, you cannot operate on return values, so the return
value is irrelevant.

In stcl/stclc mode, the command returns a list of integers
representing the saved local version numbers if you specify a
collection.  If you specify a directory, the command returns each
collection in the directory with a corresponding list of saved local
version numbers for the collection.

If a specified collection does not exists, the command throws an
error.
```

SEE ALSO

```
localversion delete, localversion restore, localversion save
```

EXAMPLES

```
The following example lists the saved local versions
of the local.sgc.loc collection.

stcl> localversion list local.sgc.loc
1 2 3

You can list all of the collections containing local versions
in a particular directory by specifying a directory instead
of a collection.

stcl> localversion list .
local.sgc.loc {1 2 3} kmlocal.sgc.loc2 3
```

localversion restore

localversion restore Command

NAME

localversion restore- Restores a saved version of the given collection

DESCRIPTION

Use the 'localversion restore' command to retrieve a local version that was previously saved for the specified collection.

Some design tools implement their own basic version management by making local copies of design objects. A local copy of a design object is referred to as a 'local version', to distinguish it from the DesignSync version, which is created in the DesignSync vault upon checkin (a vault version).

Where DesignSync manages such data either through a predefined recognition package or through a developer's Custom Type Package (CTP), DesignSync incorporates the local version number into a tag name it applies upon checkin of the object. For example, if an object's local version is 6, DesignSync applies the tag <collection_type>_6 upon check-in.

You might want to save the local version using the 'localversion save' command before you check out or populate a collection. Then, you can use the 'localversion restore' command to retrieve your local version if you later decide to use it.

Notes:

- This command only affects objects of a collection defined by the Custom Type Package (CTP). This command does not affect objects that are not part of a collection or collections that do not have local versions.
- DesignSync stores the saved local versions within the workspace directory. If you delete the workspace directory, you cannot recover the local versions.
- If you apply 'localversion restore' to an unmanaged object, the command fails.

SYNOPSIS

localversion restore <sync collection>|<sgc collection> <locint>

OPTIONS

Synchronicity Predefined Collection

```
sync collection    - Specify a Synchronicity predefined collection
                     in the format <object>.sync.<collectiontype>
                     where

                     <object> is the base name of the object, for
                     example, a cell or view name.

                     sync indicates a Synchronicity predefined
                     collection.

                     <collectiontype> is the collection name.
                     Predefined collection names include cds
                     and mw.

                     Examples of Synchronicity predefined collection
                     objects include NAND.sync.mw and cell1.sync.mw.

                     You can specify the object as a local URL
                     (starts with the file:/// protocol) or as an
                     absolute or relative path.
```

Custom Generic Collection

```
sgc collection     - Specify a custom generic collection (defined
                     in a Custom Type Package) using the format
                     <object>.sgc.<collectiontype> where

                     <object> is the base name of the object, for
                     example, a cell or view name.

                     sgc indicates a custom collection defined
                     in a Custom Type Package.

                     <collectiontype> is the collection name
                     defined in a Custom Type Package (CTP).
                     See the DesignSync Custom Type System
                     Programmer's Guide for more information.
                     An example of a custom generic collection
                     object is symbol.sgc.mytool.

                     You can specify the object as a local URL
                     (starts with the file:/// protocol) or as an
                     absolute or relative path.
```

Local Version Integer

```
locint            - Specify the integer assigned to the saved
                    local version of the collection.  If you
                    do not know the integer, use the
                    'localversion list' command to view the
                    saved local version numbers.
```

RETURN VALUE

```
In dss/dssc mode, you cannot operate on return values, so the
return value is irrelevant.

In stcl/stclc mode, the command returns the integer corresponding
to the restored local version. If a collection has no saved local
versions or if the value specified for the saved local version
number is not an integer, the command throws an error.
```

SEE ALSO

```
ctp, localversion, localversion delete, localversion list,
localversion save
```

EXAMPLES

```
The following example restores the saved local version of the
local.sgc.loc collection.  The example first lists the saved local
versions of the local.sgc.loc collection.

stcl> localversion list .
local.sgc.loc {1 2 3}
dss> localversion restore local.sgc.loc 3
3
```

localversion save

localversion save Command

NAME

```
localversion save   - Saves local version of the given collection
```

DESCRIPTION

Use the 'localversion save' command to save the current local version of a collection in preparation for fetching an alternate local version from the vault. This command does not remove the local version from your workspace. If you need a saved local version in the future, you use the 'localversion restore' command to retrieve it.

Some design tools implement their own basic version management by making local copies of design objects. A local copy of a design object is referred to as a 'local version', to distinguish it from the DesignSync version, which is created in the DesignSync vault upon checkin (a vault version).

Where DesignSync manages such data either through a predefined recognition package or through a developer's Custom Type Package (CTP), DesignSync incorporates the local version number into a tag name it applies upon checkin of the object. For example, if an object's local version is 6, DesignSync applies the tag <CollectionType>_6 upon checkin.

Depending upon the options you choose, the DesignSync 'co' and 'populate' commands can remove local versions of an object, replacing them with the requested version from the DesignSync vault. You can save the local versions using the -savelocal option to the 'co' or 'populate' commands. You can also save the local versions using the 'localversion save' command and later retrieve them using the 'localversion restore' command. The 'localversion list' command lets you view the saved local versions.

By default, check-out and populate operations on collections fail if your workspace contains a local version with a higher number than the local version being fetched. You can change the local version default behavior so that DesignSync automatically saves or removes the local versions before fetching. To change this setting, a Synchronicity administrator can use the Local Versions field on the Command Defaults pane of the SyncAdmin tool. For information, see SyncAdmin help.

Notes:

- This command only affects objects of a collection defined by the Custom Type Package (CTP). This command does not affect objects that are not part of a collection or collections that do not have local versions.

- DesignSync stores the saved local versions within the workspace directory. If you delete the workspace directory, you cannot recover the local versions.

SYNOPSIS

```
localversion save <sync collection>|<sgc collection>
```

OPTIONS

Synchronicity Predefined Collection

```
sync collection    - Specify a Synchronicity predefined collection
                     in the format <object>.sync.<collectiontype>
                     where

                     <object> is the base name of the object, for
                     example, a cell or view name.

                     sync indicates a Synchronicity predefined
                     collection.

                     <collectiontype> is the collection name.
                     Predefined collection names include cds
                     and mw.

                     Examples of Synchronicity predefined collection
                     objects include NAND.sync.mw and cell1.sync.mw.

                     You can specify the object as a local URL
                     (starts with the file:/// protocol) or as an
                     absolute or relative path.
```

Custom Generic Collection

```
sgc collection     - Specify a custom generic collection (defined
                     in a Custom Type Package) using the format
                     <object>.sgc.<collectiontype> where

                     <object> is the base name of the object, for
                     example, a cell or view name.

                     sgc indicates a custom collection defined
                     in a Custom Type Package.

                     <collectiontype> is the collection name
                     defined in a Custom Type Package (CTP).
                     See the DesignSync Custom Type System
                     Programmer's Guide for more information.
                     An example of a custom generic collection
                     object is symbol.sgc.mytool.

                     You can specify the object as a local URL
                     (starts with the file:/// protocol) or as an
                     absolute or relative path.
```

RETURN VALUE

```
In dss/dssc mode, you cannot operate on return values, so the
return value is irrelevant.

In stcl/stclc mode, the command returns an integer representing
the saved local version.

If a specified collection does not exists, the command throws
an error.
```

SEE ALSO

```
ctp, localversion, localversion delete, localversion list,
localversion restore
```

EXAMPLES

Example of Saving the Current Local Version

```
The following example saves the current local version of the
local.sgc.loc collection.  The example lists the saved local
versions of the local.sgc.loc collection.

stcl> localversion save local.sgc.loc
3
stcl> localversion list local.sgc.loc
1 2 3
```

Example of Saving Local Version of Milkyway Data

```
This example shows how localversion commands might be used for
Milkyway data.

Note: The DesignSync Milkway integration has been deprecated. This
example is meant to be used only as a reference.

In this scenario, Fadi checks out the Milkyway collection object
top_design.sync.mw to fix a defect assigned against the object,
thus fetching local version number 2 to his workspace.  He edits
the object, creating local version 3. However, he finds out
Jocelyn has already made a fix for the defect when she checked
in her local version 3. Before he checks out her local version,
he saves his local versions:
```

```
stcl> cd /home/tfadi/top_design_library
stcl> localversion save top_design.sync.mw
3

Later the team decides that Jocelyn's fix was not efficient, so
Fadi decides to retrieve his local version.

stcl> localversion list top_design.sync.mw
3
stcl> localversion restore top_design.sync.mw 3
3
```

Data Import/Export with DesignSync

exportmod

exportmod Command

NAME

```
exportmod           - Export module from a specified URL
```

DESCRIPTION

```
This command compresses the specified module into a tar file so it
can be moved to a different location, a different
category, or a different server. The command tars the entire module
contents including the module history, the module members, the
original host, port, and module URL, and references to and from the
module.

Note: Any notes, access controls, subscriptions, or mirrors
associated with the module are not exported along with the module.

The tar file that is created is stored on the server in the following
unique location:
<server-data-directory>/Export.sync/<category-path>/<module-name>.tar

Note: By providing a single, unique location for the archive file,
DesignSync avoids the possibility of overwriting the archive with a
different module of the same name. It also ensures that only one
tarred version of the of the module can exist on the server at any
given time.

The <server-data-directory> is:
 <sync_data_directory_defined_at_install_time>/<host>/<port>/server_vault

Tip: When the export is created, the output of the export command
provides the full path location to the export file. Save this
information for use with the import command.
```

Part of the moving process (export and import together) focuses on updating the hierarchical references to and from the module. This information is used when determining where the module is used (visible with the DesignSync whereused command). When the module is exported, the whereused information still identifies the original module location. When the module is imported to the new location, the hierarchical references are recreated and this new module is added to the whereused information of the referenced submodules. DesignSync does not remove, on import, the references to the old module since you are not required to delete the module.

Important: By default, the command freezes the module before beginning the exportmod and does not remove the freeze when the operation completes.

This command is subject to access controls on the server. See the ENOVIA Synchronicity Access Control Guide for details.

This command supports the command defaults system.

SYNOPSIS

```
exportmod [-[no]force] [-[no]freeze] <ServerURL>
```

ARGUMENTS

- Server URL

Server URL

```
<ServerURL>     Specifies the URL of the module. Specify the URL as
                follows:
                sync://<host>[:<port>]/Modules/[<category...>/]<module>
                or
                syncs://<host>[:<port>]/Modules/[<category...>/]<module>
                where 'sync://' or 'syncs://'
                is required, <host> is the machine on which the
                SyncServer is installed, <port> is the SyncServer
                port number (defaults to 2647/2679), [<category...>}
                is the optional category (and/or sub-category)
                containing the module, and <module> is the name of
                the module.
                For example:
                sync://serv1.abco.com:1024/Modules/ChipDesigns/Chip
```

OPTIONS

- -[no]force
- -[no]freeze

-[no]force

-[no]force	Overwrites the previous version of the exported module, if a previous version exists. -noforce does not remove the previous version. (Default) -force removes the previous version. Note: Because the name and location of the exported module is fixed based on the module and category name, only one version of the transportable module can exist at a time. For information on locating the exported module, see the Description section.

-[no]freeze

-[no]freeze	Freezes all the module branches on the server, so that no changes can be made, preserving the integrity of the information being exported. -nofreeze does not freeze module. This means changes can be made both during and after the exportmod operation. -freeze freezes the module so no changes can be made. This mode persists after the exportmod operation completes to support moving the module to a new location. (Default) Note: You can remove the module freeze using the unfreeze command.

RETURN VALUE

There is no return value. DesignSync provides status messages while the command runs. If the command fails, DesignSync returns an error explaining the failure.

SEE ALSO

importmod, freezemod, unfreezemod, mvmod

EXAMPLES

- Exporting a module

Exporting a module

```
This example creates a transportable module from an existing, in
production module to move to a new location.

dss> exportmod sync://serv1.ABCo.com:2647/Modules/Chips/chip-nx1
Beginning module export ...
sync://serv1.ABCo.com:2647/Modules/Chips/chip-nx1 : Module is frozen.
Module successfully exported.
/V6R2014Server/syncdata/serv1/2647/server_vault/Export.sync/Modules/
 Chips/chip-nx1.tar
```

exportVaults

exportVaults

NAME

```
exportVaults        - Exports DesignSync vault folders
```

DESCRIPTION

```
Use the exportVaults utility to export data from client vault folders.
See DesignSync Data Manager User's Guide: "Using the Vault
Utilities." For other export scenarios, use the ProjectSync Export
Projects feature. See ProjectSync User's Guide:"Exporting Projects."
```

import

import Command

NAME

```
import              - Fetches an object, leaving it unmanaged
```

DESCRIPTION

```
This command fetches local copies of the specified objects from the
specified vault to your current workspace. Unlike fetching with the
"co" command, imported files do not retain their association with the
vault (are no longer managed).
```

```
The "import" command can be used to switch an object's vault
association. Perform the import on the object and then run the ci
command on the  new, unmanaged, object to check it into the new
vault.

Note: The selector list can be used to select what versions to fetch.
If the select list is used, it is inherited from parent folder (the
folder into which the objects are imported). If the selector is not
appropriate for the vault from which you are importing use the
-version option to specify the version. For DesignSync objects, the
selector list will pick up tagged versions or version numbers.  For
modules, the selector list can only specify version numbers.
```

SYNOPSIS

```
import  [-force] [-version <selector>] [--]
       <argument> <object> [<object>...]
```

ARGUMENTS

- Module URL (Module-based)
- Vault URL (File-based)

Module URL (Module-based)

<module URL>	Specifies the DesignSync URL of the module for the object being imported. Specify the URL (for example: sync://srvr2.ABCo.com/Modules/Chip/chip.c;) when the object being imported is a member of a module.

Vault URL (File-based)

<vault URL>	Specifies the DesignSync vault URL for the object being imported. Specify the vault (for example: sync://system:30138/Projects/Sportster/test/runit;) when the object being imported is not a member of a module.

OBJECTS

- Module Member (Module-based)
- DesignSync File Object (File-based)

Module Member (Module-based)

```
<module member>     Specifies the module member to import. You cannot
                    import folders.
```

DesignSync File Object (File-based)

```
<DesignSync object> Specifies the file object to import. You cannot
                    import folders.
```

OPTIONS

-force

```
-force              Overwrites a local object if the object has the
                    same name as an object being imported.  When
                    -force is not specified, the default behavior is
                    to not overwrite local objects and return an
                    error message explaining why the objects were not
                    imported.
```

-version (Module-based)

```
-version <selector> Specifies the version of the objects being
                    imported.

                    If no version is specified, the default version
                    imported is the latest object version in the
                    module version specified by the module URL
                    argument.

                    Note: To use -version to specify a branch,
                    specify both the branch and version as follows:
                    '<branchtag>:<versiontag>', for example,
                    'Rel2:Latest'. You can also use the shortcut,
                    '<branchtag>:', for example "Rel2:".  If
                    you don't explicitly specify the branch
                    selector in this way, DesignSync does not
                    resolve the selector as a branch selector.
```

-version (Legacy-based)

```
 -version <selector>  Specifies the version of the objects or
                      individual member vault being imported.

                      If no version is specified, DesignSync inherits
                      the selector of the parent folder (the folder
                      into which the objects are imported).

                      Note: To use -version to specify a branch,
                      specify both the branch and version as follows:
                      '<branchtag>:<versiontag>', for example,
                      'Rel2:Latest'. You can also use the shortcut,
                      '<branchtag>:', for example "Rel2:".  If
                      you don't explicitly specify the branch
                      selector in this way, DesignSync does not
                      resolve the selector as a branch selector
```

-version (File-based)

```
 -version <selector>  Specifies the version of the objects or
                      individual member vault being imported.

                      If no version is specified, DesignSync inherits
                      the selector of the parent folder (the folder
                      into which the objects are imported).

                      Note: To use -version to specify a branch,
                      specify both the branch and version as follows:
                      '<branchtag>:<versiontag>', for example,
                      'Rel2:Latest'. You can also use the shortcut,
                      '<branchtag>:', for example "Rel2:".  If
                      you don't explicitly specify the branch
                      selector in this way, DesignSync does not
                      resolve the selector as a branch selector.
```

--

```
 --                   Indicates that the command should stop looking for
                      command options. Use this option when arguments
                      to the command begin with a hyphen (-).
```

RETURN VALUE

```
 none
```

SEE ALSO

```
co, populate, selectors
```

EXAMPLES

- Example of Importing a Specific Module Version (Module-based)
- Example of Importing a Module Member (Module-based)
- Example of Moving Files to a New Vault Associated with a Workspace (File-based)

Example of Importing a Specific Module Version (Module-based)

```
This example fetches a specific version of a module object by its
natural path.

  dss> import sync://cassini:2647/Modules/Chip;1.5 /libs/df2test/cdsinfo.tag
```

Example of Importing a Module Member (Module-based)

```
This example shows fetching a specific module member vault version
using the -version option to specify the version number.

  dss> import -version 1.3 sync://h:p/Modules/Chip;1.5\
  /libs/df2test/cdsinfo.tag
```

Example of Moving Files to a New Vault Associated with a Workspace (File-based)

```
This example performs a "switch vault" operation, where files from
one vault are imported into a work area, then checked into another
vault (the vault associated with the work area).

  dss> scd /users/jane/myworkdir
  dss> import -version Trunk sync://cassini:2647/Projects/Saturn/Rocket \
       rover.doc lander.doc
  rover.doc:   Success Imported
  lander.doc:  Success Imported

  dss> ls rover.doc lander.doc
  Time Stamp         Status  Version          Locked By   Name
  ----------         ------  -------          ---------   ----
  05/04/2000 09:24        -  Unmanaged                    rover.doc
  05/04/2000 09:24        -  Unmanaged                    lander.doc

Jane can now check these files into the vault associated with her
work area:

  dss> ci -new -nocom -keep rover.doc lander.doc
```

importmod

importmod Command

NAME

```
importmod           - Import exported module to new server location
```

DESCRIPTION

```
This command uncompresses an exported module from the tar file to the
specified location. The new module contains the full module history
of the old module, the module members, the original host, port, and
module URL information.  It also contains the hierarchical reference
information. In an additional step, you can recreate the hierarchical
references using the reconnectmod command.

Before you perform the import, you must copy the exported file to the
specific location that corresponds to the desired location on the
server. Copy the file to the following location:

<server-data-directory>/Import.sync/Modules/<category_path>/ \
<modulename>.tar

Where:
<server-data-directory> is:
  <path_to_syncdata>/<host>/<port>/

If you are also changing the name of the module, as well as the
location, rename the tar file to <newModuleName>.tar.

Note: The specified module location must be empty in order to import
the module. If there is already a module in that location, you must
remove it before performing the import.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.

This command supports the command defaults system.
```

SYNOPSIS

```
importmod[-[no]freeze] [-[no]keep] <ServerURL>
```

ARGUMENTS

- Server URL

Server URL

```
<ServerURL>     Specifies the URL of the module. Specify the URL as
                follows:
                sync://<host>[:<port>]/Modules/[<category...>/]<module>
```

```
or
syncs://<host>[:<port>]/Modules/[<category...>/]<module>
where 'sync://' or 'syncs://'
are required, <host> is the machine on which the
SyncServer is installed, <port> is the SyncServer
port number (defaults to 2647/2679), [<category...>}
is the optional category (and/or sub-category)
containing the module, and <module> is the name of
the module.
For example:
sync://serv1.abco.com:1024/Modules/ChipDesigns/Chip
```

OPTIONS

- -[no]freeze
- -[no]keep

-[no]freeze

```
-[no]freeze     Freezes all the module branches on the server after
                the import completes so any additional changes can be
               made before the module is released for normal usage.

                -nofreeze immediately releases the freeze on the
                 module after the import has completed. This means
                 changes immediately upon completion of the importmod
                 operation.

                -freeze leaves the module in a frozen state after the
                import so no changes can be made. (Default)

                Note: You can remove the module freeze using the
                unfreeze command.
```

-[no]keep

```
-[no]keep       Indicates whether DesignSync should keep or delete
                the module export file after the import is complete.

                -nokeep removes the module export file after
                 completing the import. If the import is not
                 successful, the export file is not removed,
                 regardless of how this is set. (Default)

                -keep saves the module export file after completing
                 the import.
```

RETURN VALUE

```
There is no return value. DesignSync provides status messages while
the command runs.  If the command fails, DesignSync returns an error
explaining the failure.
```

SEE ALSO

```
exportmod, mvmod
```

EXAMPLES

- Example of Importing a module

Example of Importing a module

```
This example copies a transportable module, created with the
exportmod command, changes the name of the module, and imports the
module to the new server location.

 syncmgr@serv1> cp

/usr/syncmgr/syncdata/serv1/2647/server_vault/Export.sync/Modules/Chips/Chip-
nx1.tar

/usr/syncgmr/syncdata/serv2/2647/server_vault/Import.sync/Modules/ChipDesign/
Chip-NX2.tar
 syncmgr@serv1> dssc
 dss> importmod sync://serv2.ABCo.com:2647/Modules/ChipDesign/Chip-NX2
  Beginning module import ...
  sync://serv2.ABCo.com:2647/Modules/ChipDesign/Chip-NX2 : Module is frozen.
  Module successfully imported.
```

importVaults

importVaults

NAME

```
importVaults        - Imports DesignSync vault folders
```

DESCRIPTION

```
Use the importVaults utility to import data converted using the
convertdata utility or data exported from a client vault. See
DesignSync Data Manager User's Guide: "Using the Vault Utilities."
For other import scenarios, use the ProjectSync Import Projects
feature. See ProjectSync User's Guide: "Importing Projects."
```

upload

upload Command

NAME

```
upload              - Upload/Update compressed IP stored in DesignSync
```

DESCRIPTION


```
The command allows you to upload or update a tar or gzipped tar
archive to DesignSync in an efficient manner so that, instead of
replacing the archive with the next version, DesignSync updates
only the elements within the archive file that have changed from the
previous version.

By performing a change (delta) calculation and only checking in the
changed object set, DesignSync provides both improved speed during
checkin and checkout and reduces the amount of disk space required
for storing the IP.

The user running the upload should examine the tar file to make sure
it contains none of
the following:
 o unnecessary or undesired parent directories
 o absolute path directories

These should be removed before performing the upload.

Notes:
 o The executables (binaries) for tar or gtar must be on the user's
   path in order for the command to work.

 o DesignSync also provides a graphical user interface for uploading
   IP through the DesignSync Web Interface.  For more information, see
   the DesignSync Administrator's Guide.

This command is subject to Access Controls on the server.

This command supports the command defaults system.
```

Understanding How a Temporary Directory is used for Upload

```
The compressed archive is exploded in a temporary directory and
compared against the last version, if applicable, on the server and
only the changed object set is checked in.
```

Tip: For optimal operation, DesignSync recommends that the upload directory contain at least 2.5* the size of the uncompressed archive file.

By default, this operation is performed in the temporary directory specified by the Upload_Tmp_Dir registry setting or the SYNC_TMP_DIR environment variable. If neither of these is set, DesignSync uses the /tmp directory on the repository server. For more information on setting the Upload_Tmp_Dir registry setting, or the SYNC_TMP_DIR environment variable, see the DesignSync Administrator's Guide.

You can optionally specify either a local directory or an alternate location on the server. This is especially useful for servers where you cannot control the server space consumption; specifying an alternative disk partition or performing the delta comparison locally allows you to make sure you have enough space to perform the operation. Specifying an option on the command line overrides any existing settings.

Order of Precedence for Temp Directory:

Note: DesignSync will use this order to determine which tmp directory to use for the upload operation. If there is no set value, DesignSync will check the next location on this. If there is a value set, but DesignSync is unable to use it, for example, because of incorrect write permissions, the command will fail.

1. If the -vault option is used, and -servertmpdir or -localtmpdir is specified, the value of <tmpdir> is used. If the -workspace option is specified, the workspace is used as the tmp directory.

2. If the command defaults system is used to set a value -servertmpdir or -localtmpdir, that value is used as the tmp directory.

3. If the UploadTmpDir registry setting is specified, that value is used as the tmp directory.

4. If the SYNC_TMP_DIR environment variable is set on the server machine, that value is used as the tmp directory.

5. If the TMPDIR environment variable is set on the server machine, that value is used as the tmp directory.

6. If no other values are set, DesignSync uses the /tmp directory on the server machine.

SYNOPSIS

```
upload [-branch <branchname>] [-[no]collections]
       [-[no]comment <comment>] [-[no]new]
```

```
[-report brief | normal | verbose] [-tag <tagname>]
[-vault <vaulturl> [-servertmpdir <tmpdir>] |
[-vault <vaulturl> [-localtmpdir <tmpdir>] |
[ -workspace <path>] <tarfile>
```

ARGUMENTS

- Tar file

Tar file

<tarfile>	Specify a tar or gzipped tar archive to upload or update on the server. The archive can be specified with an absolute or relative path. The file extension for the tar file must be either .tar or .tgz in order for DesignSync to recognize the file. NOTE: If the tar file contains .SYNC directories, they are automatically ignored and not checked in with the archive.

OPTIONS

- -branch
- -[no]collection
- -[no]comment
- -localtmpdir
- -[no]new
- -report (Module-based)
- -report (File-based)
- -servertmpdir
- -tag
- -vault (Module-based)
- -vault (File-based)
- -workspace

-branch

-branch <branchname>	Specifies the branch on which to place the archive. You can specify only one branch with this option. If no branch is specified, DesignSync uploads to the Trunk branch. You cannot specify a branch tag for the initial archive upload, which is always checked into the Trunk branch. For the <branchname>, specify a branch tag (for example, rel40) or branch numeric (for

example, 1.4.2) only. Do not specify a colon (:) with the branchname.

If a temp directory (other than the /tmp default) is specified for the upload, and the -branch option is used, the specified branch must already exist on the server.

The -branch option is mutually exclusive with the -new option.

-[no]collection

-[no]collection	Specifies whether the compressed package includes collections objects. For more information on collection handling, see the DesignSync Administrator's Guide. -nocollection specifies that the compressed archive does not contain collection objects. This allows the upload process to use reference mode, improving the speed of operations. (Default) -collection specifies that the compressed archive contains collection objects. The upload process will not attempt to use reference mode which would process collections incorrectly.

-[no]comment

-[no]comment ["<comment>"]	Specifies whether a text description of the upload is stored with the checked in version. -nocomment performs the upload with no comment.(Default) -comment <text> stores the value of <text> as the module comment. To specify a multi-word comment, use quotation marks ("") around the comment text.

-localtmpdir

-localtmpdir <tmpdir>	When -vault is used, the -localtmpdir option is used to specify a tmp directory path on the local (client) machine to be used for the upload operation. When the upload is completed, any objects placed in this directory during upload are removed.

-[no]new

-[no]new	Performs the initial checkin of the archive. The initial archive checkin must be performed on the Trunk branch. -nonew is used to update the archive in revision control. If the archive does not exist and -nonew is selected, the command fails. (Default) -new is used to create or update the archive. If the archive exists and the -new option is specified, the archive is updated. The -new option is mutually exclusive with the -branch option.

-report (Module-based)

-report brief | normal| verbose	Controls the amount and type of information displayed by the command. brief mode reports the newly created module version, along with the generated tag. Normal mode additionally reports a list of the changes in the archive, including: added files, removed files and changed files. Verbose mode is equivalent to normal mode.

-report (File-based)

-report brief | normal| verbose	Controls the amount and type of information displayed by the command. brief mode reports the generated tag. Normal mode additionally reports a list of the changes in the archive, including: added files, retired files and changed files. Verbose mode is equivalent to normal mode.

-servertmpdir

-servertmpdir <tmpdir>	When -vault is used, the -servertmpdir option is used to specify a tmp directory path on the repository server to be used for the upload

operation. When the upload is completed, any objects placed in this directory during upload are removed.

-tag

-tag <tag> Applies the specified tag to the data being imported. This tag can be used to get the data later, or example, when populating the archive into a workspace.

If the tag already exists it moves to the new version.

Note: An automatically generated tag, in the form Archive.<#> is also applied to the data being imported, where the initial value of # is 1, and then the number is incremented as archive is updated.

-vault (Module-based)

-vault <vaultURL> [-servertmpdir <tmpdir>] | [-localtmpdir <tmpdir>] Specify the module URL and optionally a server or local path to use as a temporary upload directory.

Specify the module URL in the format:

sync[s]://<host>:<port>/Modules/[<category>...]/<Module>

If the module does not exist, then it will be created, if the command is run with the -new option.

When you specify an alternate tmp directory for upload, you can specify a server path on on the repository server or a local path on the client system. For more information on specifying a server path, see the -servertmpdir option. For more information on specifying a local path, see the -localtmpdir option.

This option is mutually exclusive with -workspace. Either -workspace or -vault must be specified.

-vault (File-based)

-vault <vaultURL> [-servertmpdir <tmpdir>] Specify the vault URL and optionally a server or local path to use as a temporary upload

| [-localtmpdir <tmpdir>] directory.

Specify the vault URL in the format: sync[s]://<host>:<port>/[<Project>...]/<vault>

If the vault does not exist, then it will be created, if the command is run with the -new option.

When you specify an alternate tmp directory for upload, you can specify a server path on on the repository server or a local path on the client system. For more information on specifying a server path, see the -servertmpdir option. For more information on specifying a local path, see the -localtmpdir option.

This option is mutually exclusive with -workspace. Either -workspace or -vault must be specified.

-workspace

-workspace <path>

Specify an existing, unmodified workspace as a staging area to unpack the new archive, determine the changes necessary and send only the changes to the server. If this is used for an initial upload, the archive is unpacked in the workspace and the entire contents of the archive is uploaded. For the initial upload, DesignSync uses the persistent selector to determine the module/vault for checkin.

This is a performance enhancement that minimizes the server processing time needed to compute the deltas by pre-computing the deltas in the workspace.

The workspace must be owned and writable by the person running the command.

The -workspace option is mutually exclusive with -vault and -branch. The -workspace option is only supported for UNIX workspaces.

RETURN VALUE

This command does not return any TCL values. DesignSync provides status messages while the command runs. If the command fails, DesignSync returns an error explaining the failure.

SEE ALSO

```
defaults, access, ci
```

EXAMPLES

Example of Performing an Initial Upload (Module-based)

```
This example shows performing an initial upload to a module.

Note: This example has been run in normal mode, which means that each
object in the tar file is listed in the command output as it is
checked in.  For brevity, those checkin lines have been removed.

dss> upload -vault sync://qelwsun14:30126/Modules/IPWIP/FinalIP -new
 -comment "IP Finals version 1.0" FinalIP.tar

 Logging to /home/rsmith/dss_04012014_181455.log
 3DEXPERIENCE6R2022x

Beginning Check in operation...

 Checking in objects in module FinalIP%0

 Total data to transfer: 7340 Kbytes (estimate), 626 file(s), 0
collection(s)
 Checking in:
 ...

 FinalIP%0: Version of module in workspace updated to 1.2

 Finished checkin of Module FinalIP%0, Created Version 1.2

 Time spent: 10.5 seconds, transferred 0 Kbytes, average data rate
 0.0 Kb/sec

 Checkin operation finished.

 NOTE: Workspace module argument 'FinalIP%0' supplied; will tag
 'sync://serv1.ABCo.com:2647/Modules/IPWIP/FinalIP;1.2'
```

```
Beginning module tag operation on 'sync://qelwsun14:30126' ...

Tagging:     sync://serv1.ABCo.com:2647/Modules/IPWIP/FinalIP;1.2 :
Added tag 'Archive.1' to version '1.2'

Module tag operation finished on 'sync://serv1.ABCo.com:2647'.
```

Example of Specifying a Server Temporary Directory for Module Upload (Module-based)

```
This example updates an IP checked into a module.  It uses a
specified directory on the server as its temporary storage area
rather than the server default which allows you to make sure that the
space you need  for the operation is available.

Note: This example has been run in normal mode, which means that each
object in the tar file is listed in the command output as it is
populated, checked in, and tagged.  For brevity, the individual
object detail lines have been removed.

  dss> upload -vault sync://serv1.ABCo.com:2647/Modules/CustomerIP
 -servertmpdir /home/syncadmin/tmp -comment "Uploaded IP"
 ./FinalIP.tar
  Beginning Check in operation...
 Checking in: ...
 ...
 Checkin operation finished.

 Beginning Tag operation...
 ...
 Tag operation finished.
 dss>
```

Example of Specifying a Local Temporary Directory for Module Upload (Module-based)

```
This example updates an IP checked into a module.  It uses a
specified directory on the client machine as its temporary storage
area rather than the server default which allows you to make sure
that the space you need for the operation is available and reduces
processing time on the server.

Note: This example has been run in normal mode, which means that each
object in the tar file is listed in the command output as it is
populated, checked in, and tagged.  For brevity, the individual
object detail lines have been removed.

   dss> upload -vault sync://serv1.ABCo.com:2647/Modules/CustomerIP
 -localtmpdir ~/tmpfiles -comment "Uploading new version" FinalIP.tar

 Beginning Check in operation...
 Checking in: ...
 ...
 Checkin operation finished.
```

```
  Beginning Tag operation...
  ...
  Tag operation finished.
  dss>
```

Example of Performing an Upload Using a Module Workspace (Module-based)

```
 This example updates an IP checked into a module.  It uses the module
 workspace as its temporary storage area rather than the server which
 reduces the processing time needed on the server.

 Note: This example has been run in normal mode, which means that each
 object in the tar file is listed in the command output as it is
 populated and checked in.  For brevity, the individual object detail
 lines have been removed.


upload -comment "uploading IP Finals version 1.5" -workspace
       ~rsmith/MyMods/customerIP ../FinalIP.tar

  Beginning populate operation at Wed Apr 02 10:45:54 AM EDT 2014...

  Populating objects in Module          FinalIP%0
                        Base Directory  /home/rsmith/MyMods/customerIP
                        Without href recursion

  Fetching contents from selector 'Trunk:', module version '1.2'
  ... [Fetching List of Objects in Lock Mode]


  FinalIP%0 : Version of module in workspace retained as 1.2

  Finished populate of Module FinalIP%0 with base directory
/home/rsmith/MyMods/customerIP

  Time spent: 0.0 seconds, transferred 0 Kbytes, copied from local
  cache 0 Kbytes, average data rate 0.0 Kb/sec

  Finished populate operation.

  Beginning Check in operation...

 Checking in objects in module FinalIP%0

 Total data to transfer: 7102 Kbytes (estimate), 596 file(s), 0 collection(s)
  Progress: 0 Kbytes, 0 file(s), 0 collection(s), 0.0% complete
  Progress: 1 Kbytes, 0 file(s), 0 collection(s), 0.0% complete
  Progress: 4975 Kbytes, 404 file(s), 0 collection(s), 68.1% complete
  Progress: 7259 Kbytes, 596 file(s), 0 collection(s), 100.0% complete

  ... [Checking in new files, removing locks]

  FinalIP%0: Version of module in workspace updated to 1.3
```

```
  Finished checkin of Module FinalIP%0, Created Version 1.3

  Time spent: 15.7 seconds, transferred 7259 Kbytes, average data rate 463.8
Kb/sec

  Checkin operation finished.


  NOTE: Workspace module argument 'FinalIP%0' supplied; will tag
  'sync://serv1.ABCo.com/Modules/IPWIP/FinalIP;1.3'

  Beginning module tag operation on 'sync://serv1.ABCo.com:2647' ...

  Tagging:     sync://serv1.ABCo.com:2647/Modules/IPWIP/FinalIP;1.3 :
  Added tag 'Archive.2' to version '1.3'

  Module tag operation finished on 'sync://serv1.ABCo.com:2647'.
```

Example of Performing an Initial Upload (File-based)

```
 This example shows performing an initial upload to a file-based vault.

 Note: This example has been run in normal mode, which means that each
 object processed in the tar file is listed in the command output.
 For brevity, these lines have been removed.

 dss> upload -comment "IP rel 1.0 handoff" -vault
 sync://serv1.ABCo.com:2647/Projects/customerIP -new FinalIP.tar

  Operation continuing, please wait...
  sync://serv1.ABCo.com:2647/Projects/customerIP  Success Folder Made
  Logging to /home/rsmith/dss_04042014_123427.log
  3DEXPERIENCE6R2022x

  Beginning Tag operation...

  ... [List of tag files removed]

 Tag operation finished.
```

Example of Performing an Upload Using a File-Based Workspace (File-based)

```
 This example updates an IP checked into a file-based vault.  It uses a
 workspace as its temporary storage area rather than the server which
 reduces the processing time needed on the server.

 Note: This example has been run in normal mode, which means that each
 object in the tar file is listed in the command output as it is
 populated, checked in, and tagged.  For brevity, the individual
 object detail lines have been removed.
```

```
dss> upload -comment "IP rel 2.0 handoff" -workspace
~rsmith/workspaces/customerIP FinalIP.tar

 Beginning populate operation at Fri Apr 04 02:22:56 PM EDT 2014...
 ...

 Populated '/home/rsmith/workspaces/customerIP'

 Finished populate operation.

 Beginning Check in operation...
 ...

 Checkin operation finished.

 Beginning Tag operation...
 ...

 Tag operation finished.
```

Example of Specifying a Server Temporary Directory for File-based Upload (File-based)

```
This example updates an IP checked into a file-based vault.  It uses a
specified directory on the server as its temporary storage area
rather than the server default which allows you to make sure that the
space you need  for the operation is available.

Note: This example has been run in normal mode, which means that each
object in the tar file is listed in the command output as it is
populated, checked in, and tagged.  For brevity, the individual
object detail lines have been removed.

 dss> upload -vault sync://serv1.ABCo.com:2647/Projects/CustomerIP
-servertmpdir /home/syncadmin/tmp -comment "Uploaded IP"
./FinalIP.tar
  Beginning Check in operation...
 Checking in: ...
 ...
 Checkin operation finished.

 Beginning Tag operation...
 ...
 Tag operation finished.
 dss>
```

Example of Specifying a Local Temporary Directory for File-based Upload (File-based)

```
This example updates an IP checked into a file-based vault.  It uses a
specified directory on the client machine as its temporary storage
area rather than the server default which allows you to make sure
that the space you need for the operation is available and reduces
```

```
processing time on the server.

Note: This example has been run in normal mode, which means that each
object in the tar file is listed in the command output as it is
populated, checked in, and tagged.  For brevity, the individual
object detail lines have been removed.

  dss> upload -vault sync://serv1.ABCo.com:2647/Projects/CustomerIP
 -localtmpdir ~/tmpfiles -comment "Uploading new version" FinalIP.tar

 Beginning Check in operation...
 Checking in: ...
 ...
 Checkin operation finished.

 Beginning Tag operation...
 ...
 Tag operation finished.
 dss>
```

Data Replication

Data Replication System

replicate Command

NAME

```
replicate           - Data replication commands
```

DESCRIPTION

```
These commands control the data replication system. The data
replication system provides a configurable environment to
automatically setup and manage mirrors, caches, and module caches
associated with a server URL.

The replicate command and sub-commands support the command default
system.
```

SYNOPSIS

```
replicate <replicate_command> [<replicate_command_options>]

Usage: replicate [addroot, data, disable, enable, reset, rmdata,
       rmroot, showdata, showroots, setoptions ]
```

OPTIONS

Vary by command.

RETURN VALUE

Varies by command.

SEE ALSO

mirror

EXAMPLES

See specific "replicate" commands.

replicate addroot

replicate addroot Command

NAME

replicate addroot - Associates a Data Replication Root with a MAS

DESCRIPTION

This command associates a Data Replication Root (DRR) with a particular Mirror Administration Server (MAS). There are no limits to the number of DRRs that can be associated with a MAS.

The DRR is associated with the MAS using a name that must be unique across the MAS. This name is a shortcut to identify the DRR when enabling, disabling, or checking the status of the DRR. The name is also used for any auto-generated mirrors that registered to handle updates to the data replicated within the DRR.

In addition to registering the name with the MAS, the addroot command:

- creates the DRR path (specified by the -path option), if it does not already exist. If the path given to the command isn't an absolute path, DesignSync uses the current working directory with the path value appended.

- verifies that the path is suitable for storing a replication root. In order to store a replication root, the specified directory cannot contain a module cache or dynamic folder. It can contain an existing file cache.

- sets the appropriate permissions on the folder.

- creates the sub-folders needed to support data replication. The data replication system uses three folders: dynamic, for dynamic content; module_cache, for static module content; and file_cache, for static file and module member versions.

This command is subject to access controls on the server.

This command supports the command defaults system.

SYNOPSIS

```
replicate addroot -name <name> -path <rootpath> [-readmode {all|group}]
                  <serverURL>
```

ARGUMENTS

- Server URL

Server URL

```
<serverURL>      Specifies the URL of the MAS on which to store the
                 data replication root. Specify the URL as follows:
                 sync://<host>[:<port>] or
                 syncs://<host>[:<port>] where 'sync://' or 'syncs://'
                 are required, <host> is the machine on which the
                 SyncServer is installed, and <port> is the SyncServer
                 port number (defaults to 2647/2679).
                 For example: sync://serv1.abco.com:1024
```

OPTIONS

- -name
- -path
- -readmode

-name

```
-name <name>     Logical name of the DRR. This name must be unique
                 with respect to all other DRRs defined on the
                MAS. This name is also used as the name of the
                mirror that handles updates to the replications
                registered in the DRR.
```

-path

```
-path <rootpath> Specifies the path to the DRR being added.
                 If the path does not exist, and is creatable, the
                command creates it.

                 Note: The path cannot contain a module cache or
                 dynamic folder. It can contain an existing file
                 cache.
```

-readmode

```
-readmode       The read permissions set on the DRR directory. This
 [all|group]    option is valid only when SUID mode is enabled.

                -readmode all sets the read permission be readable by
               all users, the primary group of the MAS owner, and
               the MAS owner.

                -readmode group sets the read permission to be
                 readable by the primary group of the MAS owner, and
                 the MAS owner.

               Note: If SUID is not enabled for the MAS
               installation, and the "enforce SUID" option is not
               enabled on the mirror, the system will enable read
               and write modes for all. If the "enforce SUID" option
               is enabled, the command will fail. The "enforce SUID"
               option is set on the Mirrors| General Settings page
               of the DesignSync Web Interface. For information on
               setting the "enforce SUID" option for the MAS, see
               the ENOVIA Synchronicity DesignSync Administrator's
               Guide.
```

RETURN VALUE

```
Returns an empty string on success.
```

SEE ALSO

```
mirror setoptions, replicate data, replicate rmroot
```

EXAMPLES

```
This example shows the replicate addroot command and then the
```

```
replicate showroots command showing that the DRR has been created.
dss> replicate addroot -path /RepHome/repdata -name MainDRR -readmode
     all sync://data.ABCo.com
dss> replicate showroots sync://data.ABCo.com
Name         Read Mode    Path
----         ---------    ----
MainDRR      all          /RepHome/repdata
```

replicate data

replicate data Command

NAME

```
replicate data      - Replicates data on the replication root
```

DESCRIPTION

- Working with Modules Objects (Module-based)
- Working with File-Based Vaults (File-based)

```
This command adds the desired data to the specified data replication root
(DRR).

This command is subject to access controls on the server.

This command supports the command defaults system.
```

Working with Modules Objects (Module-based)

```
When you add a module to DRR, the command determines where the module
and all its submodules are located and adds them to the data
repository.

After the scripted mirrors have been registered with the MAS, the
command creates a module instance corresponding to the module
specified with the appropriate selector. This module instance will be
created in the 'dynamic' folder.
```

Working with File-Based Vaults (File-based)

```
For files-based vault data, the replicate data command calls the
mirror create command in order to create the data replication
mirror.
```

```
The base directory of the replicated data is computed from the file
path and selector information of the data, like this:
<DRR/dynamic/<ServerHost>/<port>/<leafname>/<selector>/<basedir>
```

SYNOPSIS

```
replicate data [-name <name>] -root <drr>
               [-selector <selector>[,<selector>...]] -vaulturl <URL>
                <ServerURL>
```

ARGUMENTS

- Server URL

Server URL

```
<ServerURL>      Specifies the URL of the MAS on hosting the
                 data replication root. Specify the URL as follows:
                 sync://<host>[:<port>] or
                 syncs://<host>[:<port>] where 'sync://' or 'syncs://'
                 are required, <host> is the machine on which the
                 SyncServer is installed, and <port> is the SyncServer
                 port number (defaults to 2647/2679).
                 For example: sync://serv1.abco.com:1024
```

OPTIONS

- -name (File-based)
- -root
- -selector
- -vaulturl

-name (File-based)

```
-name <name>     A user friendly name for a file-based project. The
                 name is used by the mirror create command,
```

-root

```
-root <drr>      Logical name of the DRR. This name is case
                 sensitive. If you cannot remember the name of the
                DRR, you can use the replicate showroots command to
                see a list of defined DRRs on a MAS.

                 Note: The DRR must exist on the MAS specified by the
```

```
                ServerURL argument.
```

-selector

```
-selector <list> Specifies the selector, or selector list. If no
                 selector is specified, the command will use the
                default selector, 'Trunk'.
```

-vaulturl

```
-vaulturl <URL>  Specifies the URL of the data location on the
                 server. Specify the URL as follows:
                 sync://<host>[:<port>] or
                 syncs://<host>[:<port>] where 'sync://' or 'syncs://'
                 are required, <host> is the machine on which the
                 SyncServer is installed, and <port> is the SyncServer
                 port number (defaults to 2647/2679).
                 For example: sync://serv1.abco.com:1024
```

RETURN VALUE

```
 Returns an empty string on success.
```

SEE ALSO

```
mirror create, replicate enable, replicate disable, replicate rmdata
```

EXAMPLES

- Example of Adding a Module Hierarchy to the DRR (Module-based)
- Example of Adding File-Based Data to the DRR (File-based)

Example of Adding a Module Hierarchy to the DRR (Module-based)

```
This example shows adding a module hierarchy to the DRR and the
subsequent replicate showdata command showing the module in the DRR.
Note: Although only Chip, the top level module in the hierarchy, is
specified, data replications are created for all submodules.

dss> replicate data -vaulturl
sync://data.ABCo.com:2647/Modules/ChipDesign/Chip -root MainDRR
sync://mirror.ABCo.com:2647
```

```
dss>  replicate showdata -root MainDRR sync://mirror.ABCo.com:2647
Name          Enabled  Status
Vault URL                                        Selector
Base Dir
----          -------  ------
---------                                         --------
--------
CPU%0           yes       1
sync://data.ABCo.com:2647/Modules/Components/CPU  Trunk:
9b/1e/9b1e794175a7c24bd5a329b3a5bd8d7a/CPU/Trunk/basedir
Chip%0          yes       1
sync://data.ABCo.com:2647/Modules/ChipDesign/Chip Trunk:
ef/ea/efea1173a39a7df9e42e3e8d821fd580/Chip/Trunk/basedir
ROM%0           yes       1
sync://data.ABCo.com:2647/Modules/Components/ROM  Trunk:
ee/f5/eef5d1b4b3eab3c9ec3f95b82b1b90dc/ROM/Trunk/basedir
```

Example of Adding File-Based Data to the DRR (File-based)

```
This example shows adding a file-based vault URL to the DRR and the
subsequent replicate showdata command showing the file-based vault
URL in the DRR. .

dss> replicate data -vaulturl sync://data.ABCo.com:2647/Projects/CPU
     -root MainDRR -name CPU sync://mirror.ABCo.com:2647

dss> replicate showdata -root MainDRR sync://mirror.ABCo.com:2647
Name          Enabled  Status   Vault URL
Selector       Base Dir
----          -------  ------   ---------
--------     --------
EclipseProj     yes       1
sync://data.ABCo.com:2647/Projects/EclipseProj
Trunk:Latest   9d/71/9d711cc172956426eb333ae18fb131ba/Test1/Trunk/basedir
```

replicate disable

replicate disable Command

NAME

```
replicate disable   - Disables a replicated data instance
```

DESCRIPTION

```
This command turns off updates for the specified data instance or all
replicated instances on the MAS.

This command is subject to access controls on the server.
```

```
This command supports the command defaults system.
```

SYNOPSIS

```
replicate disable -all | -name <name> -root <drr> <ServerURL>
```

ARGUMENTS

- Server URL

Server URL

```
<ServerURL>       Specifies the URL of the MAS hosting the data
                  replication root. Specify the URL as follows:
                  sync://<host>[:<port>] or
                  syncs://<host>[:<port>] where 'sync://' or 'syncs://'
                  are required, <host> is the machine on which the
                  SyncServer is installed, and <port> is the SyncServer
                  port number (defaults to 2647).
                  For example: sync://serv1.abco.com:1024
```

OPTIONS

- -all
- -name
- -root

-all

```
-all              Disables all active replication instances on the
                  DRR. This option is mutually exclusive with the
                 -name option.
```

-name

```
-name <name>      The name of the data replication to
                  disable. When the -name option is specified, only
                 the named data replication is disabled.  Any
                 referenced sub-modules continue to be updated. This
                 option is mutually exclusive with the -all option.
```

-root

```
-root <drr>       Logical name of the DRR. This name is case
                  sensitive. If you cannot remember the name of the
                 DRR, you can use the replicate showroots command to
                 see a list of defined DRRs on a MAS.
```

RETURN VALUE

```
If the command succeeds, it returns an empty string ("").  If the
command fails, it returns an appropriate error to explain the cause
of failure.
```

SEE ALSO

```
replicate enable, mirror disable
```

EXAMPLES

```
This example shows disabling all data replications on a DRR. This DRR
consists of one file-based project, and one module hierarchy.

Note: The reply from the server shows all the stopped data replications.  In
the example below, the CPU%0 and ROM%0 module instances are
referenced submodules of Chip%0.

dss> replicate disable -root MainDRR -all sync://mirror.ABCo.com
 disabling mirror 'CPU%0'
 disabling mirror 'Chip%0'
 disabling mirror 'ROM%0'
 Disabling files based mirror 'EclipseProj'
```

replicate enable

replicate enable Command

NAME

```
replicate enable    - Enables replication for a data replication
```

DESCRIPTION

```
 This command turns on updates for the specified data replication.
 If the data instance has file-based references, those are enabled by
 the replicate enable command running the mirror enable command on
 the mirror that controls the file-based updates.
```

```
This command is subject to access controls on the server.

This command supports the command defaults system.
```

SYNOPSIS

```
replicate enable -all | -name <name> -root <drr> <ServerURL>
```

ARGUMENTS

- Server URL

Server URL

```
<ServerURL>        Specifies the URL of the MAS hosting the data
                   replication root. Specify the URL as follows:
                   sync://<host>[:<port>] or
                   syncs://<host>[:<port>] where 'sync://' or 'syncs://'
                   are required, <host> is the machine on which the
                   SyncServer is installed, and <port> is the SyncServer
                   port number (defaults to 2647).
                   For example: sync://serv1.abco.com:1024
```

OPTIONS

- -all
- -name
- -root

-all

```
-all               Enables all inactive data replications on the
                   DRR. This option is mutually exclusive with the
                  -name option.
```

-name

```
-name <name>       The name of the data replication to
                   enable. This option is mutually exclusive with the
                  -all option.
```

-root

```
-root <drr>       Logical name of the DRR. This name is case
                  sensitive. If you cannot remember the name of the
                 DRR, you can use the replicate showroots command to
                 see a list of defined DRRs on a MAS.
```

RETURN VALUE

```
If the command succeeds, it returns an empty string ("").  If the
command fails, it returns an appropriate error to explain the cause
of failure.
```

SEE ALSO

```
replicate disable, mirror enable
```

EXAMPLES

```
This example shows enabling all data replications on the DRR. This
example includes a modules-based and a files based DRR.

Note: The response from the server does not display hierarchically
referenced submodules. In the example below, the Chip%0 module has
submodules CPU%0 and ROM%0 module which are enabled along with their
parent module, Chip%0, but not listed separately.

dss> replicate enable -root MainDRR -all sync://mirror.ABCo.com
 enabling mirror 'Chip%0'
 Enabling files based mirror 'EclipseProj'
```

replicate reset

replicate reset Command

NAME

```
replicate reset     - Manually updates a data replication
```

DESCRIPTION

```
This command performs a manual update on a data replication.

If the specified data replication is file-based or contains
references to a file-based sub-instance, the replicate reset
command calls the mirror reset command to update the files on the
DRR.
```

```
This command is subject to access controls on the server.

This command supports the command defaults system.
```

SYNOPSIS

```
replicate reset -name <name> -root <drr> <ServerURL>
```

ARGUMENTS

- Server URL

Server URL

```
<ServerURL>        Specifies the URL of the MAS hosting the data
                   replication root. Specify the URL as follows:
                   sync://<host>[:<port>] or
                   syncs://<host>[:<port>] where 'sync://' or 'syncs://'
                   are required, <host> is the machine on which the
                   SyncServer is installed, and <port> is the SyncServer
                   port number (defaults to 2647).
                   For example: sync://serv1.abco.com:1024
```

OPTIONS

- -name
- -root

-name

```
-name <name>       The name of the data replication to reset. The
                   reset operates recursively on the named data
                  replication.
```

-root

```
-root <drr>        Logical name of the DRR. This name is case
                   sensitive. If you cannot remember the name of the
                  DRR, you can use the replicate showroots command to
                  see a list of defined DRRs on a MAS.
```

RETURN VALUE

```
 If the command succeeds, it returns an empty string ("").  If the
command fails, it returns an appropriate error to explain the cause
of failure.
```

SEE ALSO

```
mirror reset, replicate data, replicate showdata
```

EXAMPLES

```
This example shows the reset command.
dss> replicate reset -root MainDRR -name Chip%0 sync://qelwsun14:30125
dss>
```

replicate rmdata

replicate rmdata Command

NAME

```
replicate rmdata     - Removes the data replication from the dynamic
                       folder.
```

DESCRIPTION

- Notes for Modules Objects (Module-based)
- Notes for Files-Based Objects (File-based)

```
This command removes the data replication from the 'dynamic'
folder within the DRR. The deletion cannot be undone.

Note: Although the deletion is permanent, you can recreate the
data replication using the same name using the replicate data
command.

This command is subject to access controls on the server.

This command supports the command defaults system.
```

Notes for Modules Objects (Module-based)

```
Important: To support the optimal function of the replication system,
you should use the replicate rmdata command to remove data from the
'dynamic' folder, NOT the rmmod command.  The rmmod command does not
provide the additional clean-up functionality of the replicate rmdata
command which, when reference counting is enabled, also cleans the
```

```
file cache associated with the data replication.

This command does not remove data from the module_cache folder. To
remove data from the module_cache folder, use the mcache scrub
command.

This command does not run recursively along the module
hierarchy.  You must individually remove any referenced
submodules. If you have removed a submodule, a future update to the
top-level data replication may re-create the submodule.
```

Notes for Files-Based Objects (File-based)

```
When replicate rmdata is run on a file-based replication or if a
sub-module contains links to file-based data, the replicate rmdata
command runs the mirror delete command to remove the data.
```

SYNOPSIS

```
replicate rmdata -name <name> -root <drr> <Server-URL>
```

ARGUMENTS

- Server URL

Server URL

```
<ServerURL>       Specifies the URL of the MAS hosting the
                  DRR. Specify the URL as follows:
                  sync://<host>[:<port>] or
                  syncs://<host>[:<port>] where 'sync://' or 'syncs://'
                  are required, <host> is the machine on which the
                  SyncServer is installed, and <port> is the SyncServer
                  port number (defaults to 2647).
                  For example: sync://serv1.abco.com:1024
```

OPTIONS

- -name
- -root

-name

```
-name <name>      The name of the data replication to remove.
```

-root

```
-root <drr>       Logical name of the DRR. This name is case
                  sensitive. If you cannot remember the name of the
                 DRR, you can use the replicate showroots command to
                 see a list of defined DRRs on a MAS.
```

RETURN VALUE

```
 If the command succeeds, it returns an empty string ("").  If the
command fails, it returns an appropriate error to explain the cause
of failure.
```

SEE ALSO

```
replicate data, replicate showdata, replicate rmroot, mirror delete,
```

EXAMPLES

- Example of Removing a Module Data Replication (Module-based)
- Example Showing Removing a File-Based Data Replication (File-based)

Example of Removing a Module Data Replication (Module-based)

```
This example shows removing a module data replication. The replicate
showdata command that follows shows that the Chip%0 module has been
removed.

dss> replicate rmdata -root MainDRR -name Chip%0
 sync://mirror.ABCo.com:2647

dss>  replicate showdata -root MainDRR sync://mirror.ABCo.com:2647

Name          Enabled  Status
 Vault URL                                        Selector
  Base Dir
----          -------  ------
---------                                        --------
--------
CPU%0           yes       1
  sync://data.ABCo.com:2647/Modules/Components/CPU  Trunk:
  9b/1e/9b1e794175a7c24bd5a329b3a5bd8d7a/CPU/Trunk/basedir
EclipseProj     yes       1
 sync://data.ABCo.com:2647/Projects/EclipseProj
  Trunk:Latest   9d/71/9d711cc172956426eb333ae18fb131ba/Test1/Trunk/basedir
```

Example Showing Removing a File-Based Data Replication (File-based)

```
This example shows removing a file-based vault data replication. This
shows the passed through output of the underlying mirror delete command.

dss> replicate rmdata -root MainDRR -name EclipseProj
     sync://mirror.ABCo.com:2147
Deleted empty directory '/RepHome/repdata/dynamic/9d/71/
     9d711cc172956426eb333ae18fb131ba/EclipseProj/Trunk'
Deleted empty directory '/RepHome/repdata/dynamic/9d/71/
     9d711cc172956426eb333ae18fb131ba/EclipseProj'
Deleted empty directory '/RepHome/repdata/dynamic/9d/71/
     9d711cc172956426eb333ae18fb131ba'
Deleted empty directory '/RepHome/repdata/dynamic/9d/71'
Deleted empty directory '/RepHome/repdata/dynamic/9d'
```

replicate rmroot

replicate rmroot Command

NAME

```
replicate rmroot    - removes the specified data replication root
```

DESCRIPTION

```
This command removes the specified data replication root (DRR) from the
list of registered replication roots. All the data and metadata
within the DRR are deleted along with the DRR. The deletion cannot be
undone.

Note: Although the deletion is permanent, you can recreate the DRR
using the same name.

The command does not remove any symbolic links to items in the DRR. If
users have created symbolic links, for example, workspaces still
pointing to the file cache contained within a data replication,
these need to be manually removed.

This command is subject to access controls on the server.

This command supports the command defaults system.
```

SYNOPSIS

```
replicate rmroot -root <drr> <ServerURL>
```

ARGUMENTS

- Server URL

Server URL

```
<ServerURL>       Specifies the URL of the MAS hosting the
                   DRR. Specify the URL as follows:
                  sync://<host>[:<port>] or
                  syncs://<host>[:<port>] where 'sync://' or 'syncs://'
                  are required, <host> is the machine on which the
                  SyncServer is installed, and <port> is the SyncServer
                  port number (defaults to 2647).
                  For example: sync://serv1.abco.com:1024
```

OPTIONS

- -root

-root

```
-root <drr>       Logical name of the DRR. This name is case
                   sensitive. If you cannot remember the name of the
                  DRR, you can use the replicate showroots command to
                  see a list of defined DRRs on a MAS.
```

RETURN VALUE

```
If the command succeeds, it returns an empty string ("").  If the
command fails, it returns an appropriate error to explain the cause
of failure.
```

SEE ALSO

```
replicate rmdata, replicate addroot, replicate data
```

EXAMPLES

```
This example shows removing the DRR on a MAS that has both replicated
modules and files-based data.

dss> replicate rmroot -root MainDRR sync://mirror.ABCo.com:2647
 Removing files based mirror 'EclipseProj'
 Removing the data from '/RepHome/repdata'
 Removing the dynamic scripted (autogen) mirror
 Removing the static scripted (autogen) mirror
 Removing the metadata entry for replication root 'MainDRR'
```

replicate setoptions

replicate setoptions Command

NAME

```
replicate setoptions - Sets replicate options
```

DESCRIPTION

```
This command sets the options for data replication. The available
options are described in the Options section.

This command is subject to access controls on the server.

This command supports the command defaults system.
```

SYNOPSIS

```
replicate setoptions [-defaultuser <user>] [-[no]refcount]
                     <ServerURL>
```

ARGUMENTS

- Server URL

Server URL

```
<ServerURL>       Specifies the URL of the MAS hosting the data
                  replication root. Specify the URL as follows:
                  sync://<host>[:<port>] or
                  syncs://<host>[:<port>] where 'sync://' or 'syncs://'
                  are required, <host> is the machine on which the
                  SyncServer is installed, and <port> is the SyncServer
                  port number (defaults to 2647).
                  For example: sync://serv1.abco.com:1024
```

OPTIONS

- -defaultuser
- -refcount

-defaultuser

```
-defaultuser      Species the default user for the data replication
 <user>           system. You must specify the user name with the
```

-defaultuser option. If this parameter is specified, you are prompted for the default user's password. Specifying this parameter makes 'replicate setoptions' an interactive command.

Note: You may also use the mirror setoptions command to set the defaultuser. There is only one default user stored for each MAS, regardless of whether you store the username/password with the mirror or replicate setoptions command.

-refcount

-[no]refcount — Specifies whether reference counting in the file cache is enabled or disabled for the system.

-[no]refcount specifies that references should not be counted. It is used if the site policy is not to use reference counting and the corresponding site-wide setting, set in SyncAdmin as "Enable Cache optimizations" is also disabled.

-refcount specifies that reference should be counted (Default). This allows DesignSync to maintain optimal performance by automatically removing files versions that are no longer linked to.

Note: This option is not retroactive. Any existing data continues to have the refcount setting that was in use when the data was fetched. Any subsequently fetched data uses the newly set mode.

For more information on enabling or disabling reference counting, see the ENOVIA Synchronicity DesignSync Data Manager Administrator's Guide.

RETURN VALUE

If the command succeeds, it returns an empty string (""). If the command fail, it returns an appropriate error to explain the cause of failure.

SEE ALSO

mirror setoptions, replicate addroot

EXAMPLES

```
This example shows a single replicate setoptions command that sets
the default user and disables reference counting.

dss> replicate setoptions -defaultuser admin -norefcount
sync://mirror.ABCo.com:2647
 Enter the password for the default user (admin): *****
 Processing BackupDRR
 Processing MainDRR
    Registering mirror for 'sync://mirror.ABCo.com:2647
    Done registering mirror for 'sync://mirror.ABCo.com:2647'
```

replicate showdata

replicate showdata Command

NAME

```
replicate showdata  - Lists the data replications in the DRR
```

DESCRIPTION


```
This command lists all the replication instances in the 'dynamic'
folder along with the important properties for the instance.

This command is subject to access controls on the server.

This command supports the command defaults system.
```

Notes for File-Based Objects

```
For files-based vaults, the replicate showdata  command calls the
mirror status command.
```

Understanding the Output

```
The output of the replicate showdata command can be formatted for
easy viewing (-format text) or optimized for TCL processing (-format
list).  Both viewing formats show the same information, but may have
different names. In the table below, the Column Titles column shows
the text output column header and the Property Names column shows
list output key value.
```

The replicate showdata command, by default, displays the following information:

Column Titles	Property Names	Description
Name	name	The name of the data replication.
Base Dir	basedir	Base directory of the data replication.
Status	status	Status of the DRR indicating whether the replication is functioning normally, or there are errors. o 1 (one) indicates that there are no failures in applying any updates. o 0 (zero) indicates there were failures in applying updates and the replication administrator should examine the replication log available from the DesignSync web interface.
Vault URL	vaulturl	URL of the source vault for the data replication.
Selector	selector	The selector for the data replication as supplied to the replicate data command. This varies depending on the type of href.
Enabled	enabled	Activity status of the data replication. o Yes or 1 (one) indicates the data replication is active. o No or 0 (zero) indicates the data replication is inactive.

SYNOPSIS

```
replicate showdata [-format {text|list}] [-name dataname]
                   [-report {brief|normal|verbose}] -root <drr>
```

ARGUMENTS

- -root

-root

-root <ddr>	The name of the registered DRR to examine. You can use the replicate showroots command to view the list of the DRRs.

OPTIONS

- -format
- -name
- -report

-format

```
-format text|list Determines the format of the output.

                  -format text displays a text table with headers and
                  columns. (Default) Objects are shown in
                  alphabetical order

                  -format list displays a TCL list containing
                  name/value pairs.

                  For a list of properties displayed, see the
                  "Understanding the Output" section above.
```

-name

```
-name <dataname>  The name of a data replication. If you specify
                  a name, the command returns only the results for
                 that data replication.  If no name is specified,
                 the command returns information for all the
                 data replications on the DRR.
```

-report

```
-report brief|    Determines what information is returned in the
 normal |verbose  output of the command.

                     Valid values are:
                     o brief - outputs the name of the data
                       replication and any failures. The -format
                       option is ignored for this report mode.

                     o normal - the properties list in the format
                       specified with the -format option. (Default),

                     o verbose - the properties list in the format
                       specified with the -format option. There is no
                       difference between the verbose and normal
                       reports.
```

RETURN VALUE

```
If you run the command with the '-format list' option, it returns a
TCL list. If the command fails, it returns a TCL error. For all other
options, it returns an empty string ("").
```

SEE ALSO

```
replicate showroots, replicate data, replicate rmdata
```

EXAMPLES

Example of Replicate Showdata in Text Format in Report Normal Mode

```
This example shows running replicate showdata in normal, text mode on
a DRR containing both modules and file-based vaults.  The
module is a module hierarchy consisting of top-level module Chip and
dynamically referenced submodules CPU and ROM.

dss>  replicate showdata -root MainDRR sync://mirror.ABCo.com:2647

Name          Enabled  Status
 Vault URL                                        Selector
  Base Dir
----          -------  ------
---------                                        --------
--------
CPU%0           yes       1
  sync://data.ABCo.com:2647/Modules/Components/CPU  Trunk:
  9b/1e/9b1e794175a7c24bd5a329b3a5bd8d7a/CPU/Trunk/basedir
Chip%0          yes       1
 sync://data.ABCo.com:2647/Modules/ChipDesign/Chip Trunk:
  ef/ea/efea1173a39a7df9e42e3e8d821fd580/Chip/Trunk/basedir
ROM%0           yes       1
 sync://data.ABCo.com:2647/Modules/Components/ROM  Trunk:
  ee/f5/eef5d1b4b3eab3c9ec3f95b82b1b90dc/ROM/Trunk/basedir
EclipseProj     yes       1
 sync://data.ABCo.com:2647/Projects/Test1
  Trunk:Latest   9d/71/9d711cc172956426eb333ae18fb131ba/Test1/Trunk/basedir
```

Example of Replicate Showdata in Text Format in Report Brief Mode

```
This example shows running replicate showdata in text mode with
report -brief selected using the same data set as Example 1.
```

```
dss> replicate showdata -report brief -root MainDRR
sync://mirror.ABCo.com:30125

CPU%0
Chip%0
ROM%0
EclipseProj
```

Example of Replicate Showdata in List Format in Report Normal Mode

```
This example shows running replicate showdata in -format list TCL and
-report normal mode using the same data set as Example 1.
Note: Because some of these strings exceed the line length for this
documentation, the \ character is used to show that the string does
not contain spaces.

dss> replicate showdata -format list -root MainDRR
sync://mirror.ABCo.com:2647

{name CPU%0 target sync://data.ABCo.com:2647/Modules/Components/CPU
basedir /home/RepHome/repdata/dynamic/9b/1e/\
9b1e794175a7c24bd5a329b3a5bd8d7a/CPU/Trunk/basedir selector Trunk:
version 1.3 enabled 1 reset 0 top 0 itags {} error {} seenlist \
{{sync://data.ABCo.com:2647/Modules/Components/ROM;Trunk:}} touchtime
1344447297 efile {} status 1 dynamic 1 ismodule 1} {name Chip%0 target
sync://data.ABCo.com:2647/Modules/ChipDesign/Chip  basedir
/home/RepHome/repdata/dynamic/ef/ea/efea1173a39a7df9e42e3e8d821fd580/Chip/\
Trunk/basedir selector Trunk: version 1.14 enabled 1 reset 0 top 1
itags {} error {} seenlist
{{sync://data.ABCo.com:2647/Modules/Components/CPU;Trunk:}} touchtime
1344447292 efile {} status 1 dynamic 1 ismodule 1} {name ROM%0 target
sync://data.ABCo.com:2647/Modules/Components/ROM basedir
/home/RepHome/repdata/dynamic/ee/f5/eef5d1b4b3eab3c9ec3f95b82b1b90dc/\
ROM/Trunk/basedir selector Trunk: version 1.2 enabled 1 reset 0 top 0
itags {} error {} seenlist {} touchtime 1344447301 efile {} status 1
dynamic 1 ismodule 1} {name EclipseProj target
sync://data.ABCo.com:2647/Projects/Test1 basedir
/home/RepHome/repdata/dynamic/9d/71/9d711cc172956426eb333ae18fb131ba/\
Test1/Trunk/basedir selector Trunk:Latest enabled 1 reset 0 top 0
itags {} error {} seenlist {} touchtime 0 dynamic 1 efile {} status 1
ismodule 0}
```

Example of Replicate Showdata in List Format in Report Brief Mode

```
This example shows running replicate showdata in with report -brief
selected using the same data set as Example 1.

dss> replicate showdata -report brief -root MainDRR
sync://mirror.ABCo.com:2647
```

```
CPU%0 Chip%0 ROM%0 EclipseProj
```

replicate showroots

replicate showroots Command

NAME

```
replicate showroots - Lists the registered data replication roots
```

DESCRIPTION


```
This command lists the data replication roots (DRRs) and their
properties registered on the MAS.

This command is subject to access controls on the server.

This command supports the command defaults system.
```

Understanding the Output

```
The output of the replicate showroots command can be formatted for
easy viewing (-format text) or optimized for TCL processing (-format
list).  Both viewing formats show the same information, but may have
different names. In the table below, the Column Titles column shows
the text output column header and the Property Names column shows
list output key value.

The replicate showroots command, by default, displays the following
information:

Column       Property
Titles       Names          Description
------       -----          ------------
Name         name           The name of the DRR.

Path         path           The root path to the DRR.

File cache   file_cache     The name of the file cache subdirectory.

Module cache module_cache   The name of the module cache
                            subdirectory.

Read Mode    readmode       The readmode specified when the DRR was
                            created.  The possible values are 'all'
                            and 'group.'
```

```
Dynamic      dynamic        The name of the subdirectory where the
                            dynamic data is mirrored.
```

SYNOPSIS

```
replicate showroots [-format {text|list}] [-name <rootname>]
                    [-report {brief|normal|verbose}] <ServerURL>
```

ARGUMENTS

- Server URL

Server URL

```
<ServerURL>       Specifies the URL of the MAS hosting the data
                  replication root. Specify the URL as follows:
                  sync://<host>[:<port>] or
                  syncs://<host>[:<port>] where 'sync://' or
                  'syncs://' are required, <host> is the machine on
                  which the SyncServer is installed, and <port> is
                  the SyncServer port number (defaults to 2647).
                  For example:  sync://serv1.abco.com:1024
```

OPTIONS

- -format
- -name
- -report

-format

```
-format text|list Determines the format of the output.

                  -format text displays a text table with headers and
                  columns. (Default) Objects are shown in
                  alphabetical order

                  -format list displays a TCL list containing
                  name/value pairs.

                  For a list of properties displayed, see the
                  "Understanding the Output" section above.
```

-name

```
-name <rootname>  Logical name of the DRR. This name must be unique
                  with respect to all other DRRs defined on the
                 MAS.
```

-report

```
-report brief|    Determines what information is returned in the
 normal |verbose  output of the command.

                     Valid values are:
                     o brief - outputs the name of the DRRs. The
                     -format option is ignored for this report mode.

                     o normal - the properties list, in the format
                       specified with the -format option. (Default)

                     o verbose - the properties list, in the format
                       specified with the -format option. This is
                       identical to -report normal.
```

RETURN VALUE

```
If you run the command with the '-format list' option, it returns a
TCL list. If the command fails, it returns a TCL error. For all other
options, it returns an empty string ("").
```

SEE ALSO

```
replicate addroot, replicate enable, replicate disable,
replicate rmroot
```

EXAMPLES

Example of Replicate Showroots in Text Format in Report Normal Mode

```
This example shows running replicate showroots in normal, text mode.

dss>  replicate showroots -root MainDRR sync://mirror.ABCo.com:2647

Name         Read Mode    Path
----         ---------    ----
BackupDRR    all          /home/RepBk/repdata
```

```
MainDRR      all          /home/RepHome/repdata
```

Example of Replicate Showroots in Text Format in Report Brief Mode

```
This example shows running replicate showroots in text mode with
report -brief specified using the same data set as Example 1.

dss> replicate showroots -report brief sync://mirror.ABCo.com:30125

BackupDRR
MainDRR
```

Example of Replicate Showroots in List Format in Report Normal Mode

```
This example shows running replicate showroots in -format list and
-report normal (default) mode using the same data set as Example 1.
Note: Because some of these strings exceed the line length for this
documentation, the \ character is used to show that the string does
not contain spaces.

dss> replicate showroots -format list sync://mirror.ABCo.com:2647

{name BackupDRR path /home/RepBk/repdata readmode all dynamic dynamic
module_cache module_cache file_cache file_cache} {name MainDRR path
/home/RepHome/repdata readmode all dynamic dynamic module_cache
module_cache file_cache file_cache}
```

File Cache Maintenance

cachescrubber

cachescrubber Command

NAME

```
cachescrubber       - Cleans the cache of outdated or unused files
```

DESCRIPTION

```
This command, sometimes used in conjunction with the cachetouchlinks
command, cleans your cache by deleting old or unused files.
See cachetouchlinks for more information.

The 'cachescrubber' command must be run from a Unix shell.
```

```
Note: All users have access to the cachescrubber command, although
it should only be run by the owner of the cache directory. It is
possible for users to run it on a cache directory and remove files
that should not be removed. To prevent users from inadvertently
removing files from the cache, enable SUID for caches as described in
the ENOVIA Synchronicity DesignSync Data Manager Administrator's Guide.
```

SYNOPSIS

```
cachescrubber <cache directory> [<age in days>] [-dryrun] [-noref]
              [-preds32] [-report <mode>]
```

OPTIONS

- cache directory
- age in days
- -dryrun
- -noref
- -preds32
- -report

cache directory

```
<cache directory> The directory to be scrubbed.
                  This can be an absolute or relative path.
```

age in days

```
<age in days>  Versions this age and older will be scrubbed. When
               specified, this must follow the <cache directory>
               argument. You must supply an integer value.
               This is optional when the -noref or -preds32
               options are supplied, since when using those options
               the files in the cache are not removed based on age.
               If the <age in days> argument is specified in addition
               to the -noref and/or -preds32 options, then files that
               don't meet the no reference or pre-DS3.2 criteria will
               be removed if they meet the <age in days> criteria.
```

-dryrun

```
-dryrun        Do not remove any files but report what will be removed
               when run without this option. This option can be run
               with any of the report modes.
```

-noref

```
-noref          Remove versions in the cache that have no references.
                If the <age in days> argument is also specified, the
                versions with no references will be removed regardless
                of the age criteria.
```

-preds32

```
-preds32        Remove versions put into the cache from pre-version 3.2
                DesignSync clients. If the <age in days> argument is
                also specified, the pre-3.2 versions will be removed
                regardless of the age criteria.
```

-report

```
-report         Where <mode> is brief, normal, or verbose. The 'normal'
                mode will only report warnings and errors. The 'brief'
                mode displays the same information as 'normal' mode.
                The 'verbose' mode will report all objects being
                removed. The default mode is 'normal'.
```

RETURN VALUE

```
The cachescrubber Unix command line script will return a 0 for success
and a 1 for failure.
```

SEE ALSO

```
cachetouchlinks
```

EXAMPLES

```
The following example shows how to run cachetouchlinks and
cachescrubber from a shell script. If cachetouchlinks fails,
cachescrubber will not run.

#!/bin/csh -f
#
# Touch all cache files in all workspaces listed in "workspace_files"
cachetouchlinks -file /home/syncmgr/workspace_files
if ($status != 0) then
       echo "### cachetouchlinks failed when running with"
```

```
        echo "        /home/syncmgr/workspace_files ###"
        exit 1
endif
echo "### successfully touched all cache files from workspaces listed in"
echo "       /home/syncmgr/workspace_files ###"
# If cachetouchlinks was successful, run it on all cache directories
cachescrubber /home/syncmgr/caches/sync_cache 1
if ($status != 0) then
        echo "### cachescrubber failed while cleaning sync_cache ###"
        exit 1
else
        echo "### successfully cleaned sync_cache ###"
endif
#
cachescrubber /home/syncmgr/caches/ASIC_cache 1
if ($status != 0) then
        echo "### cachescrubber failed while cleaning ASIC_cache ###"
        exit 1
else
        echo "### successfully cleaned ASIC_cache ###"
endif
cachescrubber /home/syncmgr/caches/CPU_cache 1
if ($status != 0) then
        echo "### cachescrubber failed while cleaning CPU_cache ###"
        exit 1
else
        echo "### successfully cleaned CPU_cache ###"
endif
exit 0

Note: If this shell script is run once a week, then the 'age in
days' argument could be an integer greater than 1. If the
cachetouchlinks script ran for more than 1 day, then this integer
must be greater than 1.  If 5 days were used and this script was
run once a week, this would cover the situation where the
cachetouchlinks script ran for more than 1 day (and less than 5
days). Since cachetouchlinks was run 7 days earlier, then the 'age
in days' should be less than 7. You should record the time that
cachetouchlinks ran so that the cachescrubber could be run with an
'age in days' argument that is greater than the number of days used
with cachetouchlinks.
```

cachetouchlinks

cachetouchlinks Command

NAME

```
cachetouchlinks     - Determines which files to be cleaned by the
                      cachescrubber command
```

DESCRIPTION

```
Before cleaning the cache with the cachescrubber command,
you can run cachetouchlinks on your workspace to identify which
files should be deleted. Cachetouchlinks recurses
through all specified workspaces and when it finds a local object
that is a cache type, it reads the value of the symbolic link and
"touches" the cached file that is referenced. For a collection object,
it "touches" each cached member file.

Notes:

 * The cachetouchlinks script does not consider hard links during
   processing.

 * If you unintentionally remove a cache file that is still being
   referenced, you can use the refreshcache command to recreate the
   links.

 * The cachetouchlinks command is both a UNIX command line script and
    Tcl command. Both commands have the same argument list, but
    return different values. See the Return Values section for more
    information.
```

SYNOPSIS

```
cachetouchlinks  [-checkerrors] [-dryrun] [-ignoreerrs]
 [-file <workspace-path-list>| -path <path-to-workspace-dir>]
 [-norecursive]  [-report <mode>]
```

OPTIONS

- -checkerrors
- -dryrun
- -file
- -ignoreerrs
- -path
- -norecursive
- -report

-checkerrors

```
-checkerrors     Uses the workspace metadata to verify that objects in
                 the cache state are linked to a cache.  This option
                 has a performance implication and should only be
                 used when necessary.
```

-dryrun

```
-dryrun          Only report what will be done, do not touch any
                 links. This can be run with any of the '-report'
                 modes.
```

-file

```
-file            Valid path to a file containing the list of
                 workspaces referring to caches that will be
                 cleaned. The 'list file' can be specified as an
                 absolute or relative path. The workspaces in the
                'list file' can be absolute or relative paths.
                (Entries in the file that are relative paths are
                 relative to the current working directory, rather
                 than relative to the file itself.)  When using the
                 directory list file, directories with spaces must
                 not be escaped. Leading and trailing spaces are
                 removed.

                 Workspace list file syntax:
                 one line for every workspace
                 comments have line beginning with #

                 Example workspace path list file:

                 # workspace ASIC for user larry
                 /home/larry/Projects/ASIC
                 # workspace CPU for user larry
                 /home/larry/Projects/CPU
                 # all workspaces for user frank
                 /home/frank/Projects
                 # workspace ASIC for user rizzo
                 /home/rizzo/Projects/ASIC
                 # workspace CPU for user rizzo
                 /home/rizzo/Projects/CPU

                 Note: Paths listed in the workspace path list file
                 could be absolute or relative paths. If spaces exist
                 in a path, the path should not be escaped or
                 enclosed in quotes.
```

-ignoreerrs

```
-ignoreerrs      Continue executing even if an error is encountered.
                 The default is to stop at the first error.
```

-path

```
-path            Valid path to a workspace whose corresponding cache
                 is to be cleaned. Absolute or relative paths are
```

```
accepted.  A path containing spaces must be
escaped. If run from the Unix command line, escaping
can be either using a backslash '\' preceding a
space or enclosing the path in double quotes.  When
run from an stcl shell, escaping could be using a
backslash preceding a space or enclosing the path in
double quotes or braces '{}'. When run from a dss
shell, the only escaping allowed is enclosing the
path in double quotes. A valid directory must be
specified following this option.  If this option is
not specified, cachetouchlinks will default to the
current working directory
```

-norecursive

```
-norecursive     Do not recursively process the workspace(s).
                 The default is to recursively process each
                 workspace.
```

-report

```
-report <mode>   Where mode is brief, normal, or verbose.
                 The 'normal' mode will report only warnings and
                 errors. The 'brief' mode displays the same information
                 as 'normal' mode. The 'verbose' mode will report all
                 objects that are being touched. The default is 'normal'.
```

RETURN VALUE

```
- On failure, an exception occurs (the return value is
  thrown, not returned).
- If -ignoreerrs is specified, The cachetouchlinks Tcl procedure will
  return a TCL_ERROR (1) on failure.
- On success, a TCL_OK (0) will be returned.

The cachetouchlinks Unix command line script will return a 0 for success and
a 1 for failure.
```

SEE ALSO

```
cachescrubber, refreshcache
```

EXAMPLES

```
The following example shows how you can run cachetouchlinks and
```

```
cachescrubber from a shell script. If cachetouchlinks fails,
cachescrubber will not run.

#!/bin/csh -f
#
# Touch all cache files in all workspaces listed in "workspace_files"
cachetouchlinks -file /home/syncmgr/workspace_files
if ($status != 0) then
       echo "### cachetouchlinks failed when running with"
       echo "        /home/syncmgr/workspace_files ###"
       exit 1
endif
echo "### successfully touched all cache files from workspaces listed in"
echo "       /home/syncmgr/workspace_files ###"
# If cachetouchlinks was successful, run the cachescrubber
# on all cache directories
cachescrubber /home/syncmgr/caches/sync_cache 1
if ($status != 0) then
       echo "### cachescrubber failed while cleaning sync_cache ###"
       exit 1
else
       echo "### successfully cleaned sync_cache ###"
endif
#
cachescrubber /home/syncmgr/caches/ASIC_cache 1
if ($status != 0) then
       echo "### cachescrubber failed while cleaning ASIC_cache ###"
       exit 1
else
       echo "### successfully cleaned ASIC_cache ###"
endif
cachescrubber /home/syncmgr/caches/CPU_cache 1
if ($status != 0) then
       echo "### cachescrubber failed while cleaning CPU_cache ###"
       exit 1
else
       echo "### successfully cleaned CPU_cache ###"
endif
exit 0

Note: If this shell script is run once a week, then the 'age in
days' argument could be an integer greater than 1. If the
cachetouchlinks script ran for more than 1 day, then this integer
must be greater than 1.  If 5 days were used and this script was
run once a week, this would cover the situation where the
cachetouchlinks script ran for more than 1 day (and less than 5
days). Since cachetouchlinks was run 7 days earlier, then the 'age
in days' should be less than 7. You should record the time that
cachetouchlinks ran so that the cachescrubber could be run with an
'age in days' argument that is greater than the number of days used
with cachetouchlinks.
```

refreshcache

refreshcache Command

NAME

```
refreshcache        - Refreshes a workspace, re-establishing cache links
                      to point to either a new cache location or to new
                      cache names
```

DESCRIPTION

```
This command traverses the workspace directory hierarchy in search of
objects that exist in the shared cache mode and recreates the link to the
cache.

This command can be used when moving a cache to re-establish links to
the new cache directory.  The cache file naming mechanism changed in
DesignSync version 3.2 to address consistency and reliability issues.
If pre-version 3.2 cache links still exist, the refreshcache command can be
used to re-establish links to cache files using the new cache file name
format. It is beneficial to re-establish links to cache files using the
new cache name file format since there have been consistency and reliability
issues addressed with the new cache naming. Links to the old cache file
names
will still work.

If you only have the latest version of any branch populated, first
'populate -recursive -reference -unifystate' to replace any cache links
with DesignSync references. Then 'populate -recursive -share -unifystate'
to recreate the cache links, which will now lead to cached files in the
new cache location.  This will also replace pre- DS 3.2 style cache links
with the newer cache link format.

Note that the DesignSync reference state is only intended to be temporary.
DesignSync references do not exist on disk, so tools requiring actual data
will not work properly with DesignSync references.

The refreshcache command is useful when non-latest versions exist in the
workspace since it recreates a new link to the current version in the
workspace.
```

SYNOPSIS

```
refreshcache [-continue_on_error] [-dryrun]
             [ -file <path> | -workspace <path>] [-norecursive]
             [-preds32] [-verbose]
```

OPTIONS

- -continue_on_error
- -dryrun
- -file

-continue_on_error

```
-continue_on_error  If an error is encountered, do not exit, but
                    continue processing workspace(s). By default, the
                    refreshcache command will exit on encountering an
                    error.
```

-dryrun

```
-dryrun             Only report what will be done, do not refresh any
                    links.
```

-file

```
-file               A full or relative path to a file containing a
                    list of workspaces to be refreshed.  Either this
                    option or the '-workspace' option must be
                    supplied, but not both. The format of this file
                    is one line for every workspace. A line beginning
                    with a '#' is a comment.
```

-norecursive

```
-norecursive        Do not recursively process the workspace(s).
```

-preds32

```
-preds32            The cache file naming mechanism changed in
                    DesignSync Version 3.2 to address consistency
                    and reliability issues.  It is beneficial to
                    re-establish links to the new cache file
                    names. Using this option will only select links
                    that are currently pointing to the old cache
                    file name format and re-establish them so that
                    they are linking to files using the new cache
                    file name format.

                    This can be useful if a user already has a
```

```
                    workspace with a mix of links pointing to cache
                    files with the old naming convention and the new
                    naming convention. This way it would not waste time
                    refreshing the links to files in the new name format.
```

-workspace

```
 -workspace         The full path or relative path to workspace to be
                    refreshed. Either this option or the '-file'
                    option must be supplied, but not both.
```

-verbose

```
 -verbose           Extra report processing - reports object being
                    processed, version id, object type and old cache
                    file links. The old cache file links are reported
                    with -preds32 option.
```

RETURN VALUE

```
If the refresh is successful, returns an empty string. If the refresh
fails, returns an appropriate error.
```

SEE ALSO

```
cachescrubber, cachetouchlinks, populate
```

EXAMPLES

- Example Showing a Dry Run
- Example Showing Updating Pre-3.2 Cache Files
- Example Showing Updating Workspaces From a File

Example Showing a Dry Run

```
Reports all cache links in the asic hierarchy that will be refreshed,
but doesn't refresh any:

  stcl> refreshcache -workspace /home/larry/Projects/asic -dryrun
```

Example Showing Updating Pre-3.2 Cache Files

```
Add the -preds32 option and it will report all links that point to
cache files using the old naming convention, which will be
refreshed if the -dryrun option is not specified.

Re-establishes cache links in workspace /home/larry/Projects/asic:

  stcl> refreshcache -workspace /home/larry/Projects/asic

Can use the -preds32 option to only refresh cache links that were
linked to files before version 3.2. These links are pointing to cache
files that used the old naming convention.

First report what will be refreshed.

  stcl> refreshcache -workspace /home/larry/Projects/asic -preds32 -dryrun

Re-establishes cache links:

  stcl> refreshcache -workspace /home/larry/Projects/asic -preds32
```

Example Showing Updating Workspaces From a File

```
Reports all cache links in all workspaces listed in file
'/home/larry/workspace_list', which will be refreshed, but doesn't
refresh any.

The workspace list file might look like follows:

    # comment - ASIC project
    /home/larry/Projects/asic
    # CPU project
    /home/larry/Projects/cpu

  stcl> refreshcache -file /home/larry/workspace_list -dryrun

Re-establishes cache links:

  stcl> refreshcache -file /home/larry/workspace_list
```

Caching Objects

caching

caching Command

NAME

```
caching             - Caching behavior commands
```

DESCRIPTION

```
These commands provide a way to view and control the caching behavior
of DesignSync objects; excepting or including intellectual property
from the default caching.
```

SYNOPSIS

```
caching <caching_command>

Usage: caching disable|caching enable|caching list|caching status
```

ARGUMENTS

```
Server URL
```

RETURN VALUE

```
Various by command.
```

SEE ALSO

```
caching disable, caching enable, caching list, caching status
,
```

EXAMPLES

```
See specific command.
```

caching disable

caching disable Command

NAME

```
caching disable     - Disables object caching for server URLs
```

DESCRIPTION

```
This command disables caching for specific objects specified by
```

server URLs.

When object caching is disabled, the caching property of the object URL is set to zero (0). The value of this property may be viewed with the caching status command.

Note: Clients with this feature can disable the local caching functionality for objects on an older (pre-3DEXPERIENCE 2016x) server version, but older clients cannot use this feature on newer clients. Servers accepting commands from older clients can be set up to refuse enable/disable caching requests from older clients. For more information, see the DesignSync Data Manager Administrator's Guide.

If the object for which caching is being disabled were already loaded into a cache, those caches are not automatically removed, however attempts to update the cache, for example with cancel, ci, populate, or co, will fail.

This command is subject to access controls on the server.

SYNOPSIS

```
caching disable <SyncURL>[ <SyncURL>...]
```

ARGUMENTS

- URL

URL

```
<syncURL>  Specifies the option Sync URL as follows:
           sync://<host>[:<port>]/[<path>] or
           syncs://<host>[:<port>]/[<path>]
           where 'sync://' or 'syncs://' are required, <host> is the
           machine on which the SyncServer is installed, and <port>
           is the SyncServer port number (defaults to 2647/2679).
           And the path is the server path to the desired object.
           For example, all of these are valid syncURLs:
           sync://apollo.spaceco.com:2647
           sync://apollo.spaceco.com:2647/Modules
           sync://apollo.spaceco.com:2647/Modules/Blueprints/FuelCell2
           sync://apollo.spaceco.com:2647/Projects/SpaceShuttle
```

RETURN VALUE

In stcl/stclc mode, two lists are returned, where the first list is non-empty if at least one object was successfully processed,

```
and the second list is non-empty if at least one object failed.
If the command failed to run, DesignSync returns an appropriate error
explaining the failure.
```

SEE ALSO

```
caching enable, caching list, caching status, url getprop, url setprop
```

EXAMPLES

- Example Showing Disabling cachability for an object

Example Showing Disabling cachability for an object

```
This example shows disabling the caching for a specific object and
verifying that the cachability was disabled using the status command,
which returns a status of zero (0).

dss> caching disable sync://srv1.ABCo.com:2647/Modules/ChipDesign/Chip
{Objects succeeded (1)} {}

dss> caching status sync://srv1.ABCo.com:2647/Modules/ChipDesign/Chip
0
```

caching enable

caching enable Command

NAME

```
caching enable      - Enables object caching for server URLs
```

DESCRIPTION

```
This command enables caching for specific objects specified by
server URLs.

When object caching is enabled, the caching property of the object
URL is set to one (1). The value of this property may be viewed with
the caching status command.

Note: Clients with this feature can enable the local caching
functionality for objects on an older (pre-3DEXPERIENCE 2016x) server
version, but older clients cannot use this feature on newer
clients. Servers accepting commands from older clients can be set up
to refuse enable/disable caching requests from older clients.  For
```

```
more information, see the DesignSync Data Manager Administrator's
Guide.

This command is subject to access controls on the server.
```

SYNOPSIS

```
caching enable <SyncURL>[ <SyncURL>...]
```

ARGUMENTS

- URL

URL

```
<syncURL>  Specifies the option Sync URL as follows:
           sync://<host>[:<port>]/[<path>] or
           syncs://<host>[:<port>]/[<path>]
           where 'sync://' or 'syncs://' are required, <host> is the
           machine on which the SyncServer is installed, and <port>
           is the SyncServer port number (defaults to 2647/2679).
           And the path is the server path to the desired object.
           For example, all of these are valid syncURLs:
           sync://apollo.spaceco.com:2647
           sync://apollo.spaceco.com:2647/Modules
           sync://apollo.spaceco.com:2647/Modules/Blueprints/FuelCell2
           sync://apollo.spaceco.com:2647/Projects/SpaceShuttle
```

RETURN VALUE

```
In stcl/stclc mode, two lists are returned, where the first list
is non-empty if at least one object was successfully processed,
and the second list is non-empty if at least one object failed.
If the command failed to run, DesignSync returns an appropriate error
explaining the failure.
```

SEE ALSO

```
caching disable, caching list, caching status
```

EXAMPLES

- Example Showing enabling cachability for an object

Example Showing enabling cachability for an object

```
This example shows enabling the caching for a specific object and
verifying that the cachability is enabled using the status command
which returns a status of one (1).

dss> caching enable sync://srv1.ABCo.com:2647/Modules/ChipDesign/Chip
{Objects succeeded (1)} {}

dss> caching status sync://srv1.ABCo.com:2647/Modules/ChipDesign/Chip
1
```

caching list

caching list Command

NAME

```
caching list        - Displays the cache status for folders
```

DESCRIPTION

- Understanding the Output

```
Displays a list of subfolders and/or parent folders for a specified vault
that have an explicitly set cache status. An explicitly set cache status
indicates
that caching is either enabled (on) or disabled (off) for the
folder. Folders that inherit their state from their parents are not
displayed.

This command is subject to access controls on the server.
This command respects the command defaults setting.
```

Understanding the Output

```
The output of the caching list command can be formatted for
easy viewing (-format text) or optimized for TCL processing (-format
list).  Both viewing formats show the same information, but may have
different names. In the table below, the Column Titles column shows
the text output column header and the Property Names column shows
list output key value.

The caching list command, by default, displays the following
information:
```

Column Titles	Property Names	Description
Caching Status	status	Caching status for the folder. o Enabled - In text mode, if caching is active for folder, it displays Enabled. In list mode, it displays 1. o Disabled - In text mode, if caching is inactive for the folder, it displays Disabled. In list mode, it displays 0.
Path	path	The server path to the folder.

SYNOPSIS

```
caching list [-disabled]  [-down] [-enabled]  [-format list | text]
                  [-up] <argument>
```

ARGUMENTS

- {} URL

{} URL

<SyncURL>	Specifies the DesignSync vault URL. The command examines either backwards (up) or forwards (down) to determine whether the folder has an explicitly set cache status. The vault is specified as: sync://<host>[:<port>]/[<path>] or syncs://<host>[:<port>]/[<path>] where 'sync://' or 'syncs://' are required, <host> is the machine on which the SyncServer is installed, and <port> is the SyncServer port number (defaults to 2647/2679). And the path is the server path to the desired object. For example, all of these are valid syncURLs: sync://serv1.abco.com:2647 sync://serv1.abco.com:2647/Modules sync://serv1.abco.com:2647/Modules/ChipDesigns sync://serv1.abco.com:2647/Projects/SharedLibraries

OPTIONS

{}

```
-disabled             Show all of the folders that have caching
                       explicitly disabled.
                      If this option is specified with -enabled,
                       the command shows folders that have either
                       enabled or disabled status specified,
                       which is the same as specifying neither option.
```

-down

```
-down                 Show all of the folders below the selected
                       Sync URL argument.
                      If this option is specified with -up, the
                      command shows folders up and down from the
                      specified Sync URL, which is the same as
                      specifying neither option.
```

-enabled

```
-enabled            Show all of the folders that have caching
                     explicitly enabled.
                    If this option is specified with -disabled,
                     the command shows folders that have either
                     enabled or disabled status specified,
                     which is the same as specifying neither option.
```

-format

```
-format text|list Determines the format of the output.

                  -format text displays a text table with headers and
                  columns. (Default) Objects are shown in
                  alphabetical order
```

```
                    -format list displays a TCL list containing
                    name/value pairs.

                    For a list of properties displayed, see the
                    "Understanding the Output" section above.
```

-up

```
 -up                      Show all of the folders below the selected
                             Sync URL argument.
                           If this option is specified with -up, the
                           command shows folders up and down from the
                           specified Sync URL, which is the same as
                           specifying neither option.
```

RETURN VALUE

```
 Returns an empty string when -format text is used. Returns a TCL list
 as described in the Understanding the Output section when -format list
 is used.
```

SEE ALSO

```
 caching disable, caching enable, caching status
```

EXAMPLES

- Example of Listing the Cache Status using Text Formatting
- Example of Listing the Cache Status using List Formatting

Example of Listing the Cache Status using Text Formatting

```
 This example shows listing the status of caching for all folders above
  and below the specified vault folder.
  Note that in this example, the specified module is not displayed
  because it inherits it's state from the parent category,
"/Modules/ChipDesigns"

 stcl> caching list sync://serv1.ABCo.com:2647/Modules/ChipDesigns/NXZ-45
  Caching Status    Path
  --------------           ----
  Enabled              /Modules
  Disabled             /Modules/TradeSecrets
  Enabled              /Modules/ChipDesigns
  Disabled             /Modules/ChipDesigns/NXZ-45/Proprietary
  Enabled             /Modules/ChipDesigns/NXZ-45/Propertary/ReadyForRelease
```

Example of Listing the Cache Status using List Formatting

```
 This example shows listing the status of caching for all folders below
 the specified vault folder.
Stcl> caching list -down â€"format list
sync://serv1.ABCo.com:2647/Modules/ChipDesigns/NXZ-45

{
   {path /Modules/ChipDesigns/NXZ-45/Proprietary status 0}
   {path /Modules/ChipDesigns/NXZ-45/Proprietary/ReadyForRelease status 0}
```

caching status

caching status Command

NAME

```
 caching status      - Displays caching status of server URLs
```

DESCRIPTION

```
 Displays the caching status (on or off) of the object URL.  URLs can
 be explicitly excluded from the cache to protect access to the file
 and comply with intellectual property protection needs.
```

SYNOPSIS

```
 caching status <SyncURL>
```

ARGUMENTS

- URL

URL

```
 <syncURL>  Specifies the option Sync URL as follows:
            sync://<host>[:<port>]/[<path>] or
            syncs://<host>[:<port>]/[<path>]
            where 'sync://' or 'syncs://' are required, <host> is the
            machine on which the SyncServer is installed, and <port>
            is the SyncServer port number (defaults to 2647/2679).
            And the path is the server path to the desired object.
            For example, all of these are valid syncURLs:
            sync://serv1.abco.com:2647
```

```
sync://serv1.abco.com:2647/Modules
sync://serv1.abco.com:2647/Modules/Blueprints/FuelCell2
sync://serv1.abco.com:2647/Projects/SpaceShuttle
```

RETURN VALUE

```
Returns a value of zero (0) if the object can not be cached, or one
(1) if the object is able to be cached.
```

SEE ALSO

```
caching disable, caching enable, caching list
```

EXAMPLES

Example Showing the cachability status for an object

```
This example shows enabling/disabling the caching for a specific
object and verifying that the cachability is enabled using the status
command, which returns a zero (0) if cachability is disabled or one
(1) if cachability is enabled.

dss> caching enable sync://srv1.ABCo.com:2647/Modules/ChipDesign/Chip
{Objects succeeded (1)} {}

dss> caching status sync://srv1.ABCo.com:2647/Modules/ChipDesign/Chip
1
dss> caching disable sync://srv1.ABCo.com:2647/Modules/ChipDesign/Chip
{Objects succeeded (1)} {}

dss> caching status sync://srv1.ABCo.com:2647/Modules/ChipDesign/Chip
0
```

Mirror System

mirror Commands

NAME

```
mirror              - Mirror management commands
```

DESCRIPTION

```
The mirror commands allow you to create, view, edit, and
check the status of mirrors. You also can administer mirrors
using the DesignSync WebUI.
```

SYNOPSIS

```
mirror <mirror_command> [<mirror_command_options>]

Usage: mirror [create|delete|disable|edit|enable|get|getoptions|
               isenabled|ismirror|list|rename|requeue|reset|setoptions|
              status|wheremirrored]
```

OPTIONS

```
Vary by command.
```

RETURN VALUE

```
Varies by command.
```

SEE ALSO

```
mirror create, mirror delete, mirror disable, mirror edit,
mirror enable, mirror get, mirror getoptions, mirror isenabled,
mirror ismirror, mirror list, mirror rename, mirror requeue,
mirror reset, mirror setoptions, mirror status,
mirrorsetdefaultuser, mirror wheremirrored
```

mirror

mirror Commands

NAME

```
mirror              - Mirror management commands
```

DESCRIPTION

```
The mirror commands allow you to create, view, edit, and
check the status of mirrors. You also can administer mirrors
using the DesignSync WebUI.
```

SYNOPSIS

```
mirror <mirror_command> [<mirror_command_options>]

Usage: mirror [create|delete|disable|edit|enable|get|getoptions|
               isenabled|ismirror|list|rename|requeue|reset|setoptions|
              status|wheremirrored]
```

OPTIONS

```
Vary by command.
```

RETURN VALUE

```
Varies by command.
```

SEE ALSO

```
mirror create, mirror delete, mirror disable, mirror edit,
mirror enable, mirror get, mirror getoptions, mirror isenabled,
mirror ismirror, mirror list, mirror rename, mirror requeue,
mirror reset, mirror setoptions, mirror status,
mirrorsetdefaultuser, mirror wheremirrored
```

mirror create

mirror create Command

NAME

```
mirror create       - Creates a mirror
```

DESCRIPTION

- Using Mirror Create with Modules (Module-based)

```
This command creates a mirror. When a password is required, you are
prompted and the command becomes interactive.

Note: When you create a mirror, submirrors for referenced data are
created automatically.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.
```

Using Mirror Create with Modules (Module-based)

```
When the mirror create command is run for a module, the mirror
directory must either be empty or the vault set on the mirror
directory must match that provided by the mirror create command.
This allows a user to manually populate a mirror directory prior to
registering it with the mirror system.

Mirrors can be created for modules but a workspace is not allowed to
link to it. Thus, a set mirror and a ci or populate with the mirror
fetch state are not allowed with modules.

Note: If you have two hrefs to a sub-module in a hierarchy, you
      cannot create a second sub-mirror to the second sub-module.
```

SYNOPSIS

```
mirror create [-cachedir <path>] [-cachelinktype hard|soft]
              [-category <category>] [-description <description>]
              [-[no]enable] [-fetchstate get|share]
              [-hrefmode<static|dynamic|normal>] [-MASuser <user>]
              [-mcachemode link|server] -mirrordir <mirrorDir>
              -name <name> [-notify<email_list>]
              [-parentname <parent_mirror>] [-PMASuser <user>]
              [-primaryserver <serverURL>] [-[no]recursive]
              [-RSuser <user>] [-script <TCL_script>]
              [-selector <list>] [-type normal|primary|secondary]
              -vaultURL <vaultURL> <serverURL>
```

ARGUMENTS

- Server URL

Server URL

```
serverURL       Specifies the URL of the MAS where the mirror will be
                created. Specify the URL as follows:
                sync://<host>[:<port>] or
                syncs://<host>[:<port>] where 'sync://' or 'syncs://'
                are required, <host> is the machine on which the
                SyncServer is installed, and <port> is the SyncServer
                port number (defaults to 2647/2679).
                For example:  sync://serv1.abco.com:1024
```

OPTIONS

-cachedir

```
-cachedir          The -cache option allows you specify the path to the
 <path>            mirror specific file cache. If you update mirrors
                   with the share state and no file cache is specified,
                  the default cache or project caches are used. The
                  default cache is determined by the MUP's registry
                  files which include the server's registry files and
                  the MirrorRegistry.reg file.
```

-cachelinktype

```
-cachelinktype     The -cachelinktype option indicates whether creating
 hard|soft         a mirror with cache links uses hard or soft
                   (symbolic) links in the mirror. This option is
                  ignored if the -fetchstate get option is used.

                   "-cachelinktype hard" populates the mirror with hard
                   links to the cache.  Hard links require that mirror
                   and the cache be on the same disk partition. If the
                   system cannot create hard links to the cache, it
                   will switch automatically to creating soft links to
```

the cache.

"-cachelinktype soft" populates the mirror with symbolic links to the cache. (Default)

Note: This option does not use the registry settings that determine whether a hard link or soft link should be used. The default setting for this option is to use soft links.

-category

-category — An optional parameter to assign a category (arbitrary string) to a mirror. All mirrors belonging to a category can be selected by using the -category parameter of 'mirror list'. When mirror definitions are automatically generated as the result of encountering references, they will inherit the category of their parent mirror. A category must be composed of the following set [A-Za-z0-9_/-.] or a space.

-description

-description — An optional description of the mirror. A description must be composed of the following set [A-Za-z0-9_/-.] or a space.

-[no]enable

-[no]enable — Determine if the mirror should be enabled when it is created. The default is enable.

-fetchstate

-fetchstate get|share — An optional mode to indicate whether the mirror should be populated with local copies, or populated with file cache links.

"-fetchstate get" populates the mirror with local copies. (Default)

"-fetchstate share" populates the mirror with links the cache. You can use the -cachelinktype to specify whether the links should be soft links (symbolic links) or hard links.

-hrefmode (Module-based)

```
-hrefmode       An optional mode to enable the selection of a
                hierarchical reference mode when recursively
                populating the mirror.
                Valid values are:
                o normal (default)
                o static
                o dynamic

                This option is only meaningful when <vaultURL>
                refers to a non-legacy module.

                You can use the -hrefmode with -nomodulerecursive.
                The recursive nature of the mirror can be changed
                using the mirror edit command to recursively populate
                a module.
```

-MASuser

```
-MASuser        The name of the user to establish communication from
                the repository server (RS) to the mirror
                administration server (MAS). When this parameter is
                omitted, the default user (mirror setoptions
                -defaultuser) is used for this connection. If the
                default user is not set, an error will be thrown.
                If the 'mirror setoptions -enforcedefaultuser' option
                is set on the MAS and this parameter is specified, an
                error will be thrown. If this parameter is specified,
                the user will be prompted for the corresponding
                password. Therefore, specifying this parameter makes
                the 'mirror create' command an interactive command.
```

-mcachemode (Module-based)

```
-mcachemode
 link|server     When using the scripted mirror capability to populate
                 an auto-generated mirror, this option specifies:
                 o link - Attempt to create mcache links to referenced
                   submodules by searching the mirror directory
                  supplied with the scripted/autogen mirror.
                 o server - Fetch the submodules from the server.

                 When populating a normal mirror, this option
                 specifies:
                 o link - Attempt to create mcache links to
                  referenced submodules by searching using the
                   default mcache paths defined in the registry.
```

```
    o server - Fetch the submodules from the server.
```

-mirrordir (Module-based)

```
-mirrordir      The pathname of the mirror directory.
                Note: This parameter is required.

                The pathname must be unique with respect to
                all other mirrors defined on the mirror administration
                server (MAS).

                If this pathname contains a relative path with
                respect to the stcl's cwd, it is converted to an
                absolute pathname before the name is passed to the
                mirror administration server.

                If the mirror directory pathname contains a symbolic
                link (UNIX only), the user controls whether the
                symbolic link is resolved or used as is, with the
                EnableRealMirrorPaths registry key.

                The '~' home directory character is allowed (UNIX
                only).

                The directory pathname must be relevant on the same
                LAN as the mirror administration server.

                The directory name can be prefixed with 'file://' if
                an absolute pathname is specified.

                The path of a mirror directory can be shared by other
                mirrors defined on the MAS or used for multiple
                modules. Since the base directory for multiple
                modules can be the same, the base directory can work
                as a mirror directory.

                You cannot use a mirror directory for a module mirror
                that is already used for a legacy module or a
                DesignSync vault.
```

-mirrordir (Legacy-based)

```
-mirrordir      The pathname of the mirror directory.
                Note: This parameter is required.

                The pathname must be unique with respect to
                all other mirrors defined on the mirror administration
                server (MAS).

                If this pathname contains a relative path with
                respect to the stcl's cwd, it is converted to an
```

absolute pathname before the name is passed to the mirror administration server.

If the mirror directory pathname contains a symbolic link (UNIX only), the user controls whether the symbolic link is resolved or used as is, with the EnableRealMirrorPaths registry key.

The '~' home directory character is allowed (UNIX only).

The directory pathname must be relevant on the same LAN as the mirror administration server.

The directory name can be prefixed with 'file://' if an absolute pathname is specified.

The mirror directory path and the relative path of the legacy module must be unique. You cannot use a directory for a legacy module that is already in use by another mirror.

-mirrordir (File-based)

-mirrordir The pathname of the mirror directory.
Note: This parameter is required.

The pathname must be unique with respect to all other mirrors defined on the mirror administration server (MAS).

If this pathname contains a relative path with respect to the stcl's cwd, it is converted to an absolute pathname before the name is passed to the mirror administration server.

If the mirror directory pathname contains a symbolic link (UNIX only), the user controls whether the symbolic link is resolved or used as is, with the EnableRealMirrorPaths registry key.

The '~' home directory character is allowed (UNIX only).

The directory pathname must be relevant on the same LAN as the mirror administration server.

The directory name can be prefixed with 'file://' if an absolute pathname is specified.

The mirror directory path and the relative path of the mirror must be unique. You cannot use a directory that is already in use by another mirror.

-name

-name

Logical name of the mirror. This name must be unique with respect to all other mirrors defined on the mirror administration server (MAS). When mirror definitions are automatically generated as the result of encountering references, the new mirror names should reflect the point in the mirror hierarchy where the reference was encountered. The hierarchical delimiter used must be a slash, '/'.
EXAMPLE: If the mirror 'liba' uses mirror directory /home/libs/liba and during the initial populate a reference is encountered at the /home/libs/liba/iocells/scancells directory, a new mirror called 'liba/iocells/scancells' will be created. Names must be composed of the following set [A-Za-z0-9_/-.]. Mirror names cannot begin with a dash (-).

When -name is used without the -parentname argument, the name parameter cannot be the name of an existing mirror. However, when used in combination with the -parentname argument, the name parameter can refer to an existing mirror. In this case, the mirror directory, vault URL, and selector are validated and if any of these parameters is incorrect, an error is thrown. When -name and -parentname are used in combination, all other parameters for the mirror create command are used only for validation.

The -name value is required.

-notify

-notify

A comma-separated or space-separated list of email addresses and/or user names to send email to whenever the mirror generates notifications. The defaultnotifylist (see 'mirror setoptions') will be internally appended to this list. If the defaultnotifylist has not been set then this list will default to the email address in the user's profile for the user executing this command on the MAS. If user names are specified, the users must have user profiles on the MAS servers.

-parentname

-parentname

When you are creating a submirror, the logical name of the parent mirror. The parent mirror must already exist. See the -name argument description, above, for

information on how the -name and -parentname arguments interact.

-PMASuser (File-based)

-PMASuser The name of the user to establish communication from the mirror administration server (MAS) to the primary mirror server (PMAS). If the -type parameter is not set to "secondary", this parameter is silently ignored. When the -type is set to "secondary" then if this parameter is omitted, the default user (mirror setoptions -defaultuser) is used for this connection. If the default user is not set, an error will be thrown. If the 'mirror setoptions -enforcedefaultuser' option is set on the MAS and this parameter is specified, an error will be thrown. If this parameter is specified, the user will be prompted for the corresponding password. Therefore, specifying this parameter makes the 'mirror create' command an interactive command.

-primaryserver (File-based)

-primaryserver Specifies the URL of the SyncServer hosting the primary mirror (PMAS). If the -type parameter is not set to "secondary", this parameter is silently ignored. Specify the URL as follows: sync://<host>[:<port>] or syncs://<host>[:<port>] where 'sync://' or 'syncs://' are required, <host> is the machine on which the PMAS is installed, and <port> is the PMAS port number (defaults to 2647) For example" -primaryserver sync://serv1.abco.com:1024

-[no]recursive (Module-based)

-[no]recursive Determines whether to mirror the module's contents only or the entire module hierarchy.

-recursive specifies that the mirror populates a module's contents and the contents of all its sub-modules recursively. (Default)

-norecursive specifies that the mirror populates a module without processing the module hierarchy.

-RSuser

-RSuser	The name of the user to establish communication from the mirror administration server (MAS) to the repository server (RS). This User must have a user profile on the RS. When this parameter is omitted, the default user (mirror setoptions -defaultuser) is used for this connection. If the default user is not set, an error will be thrown. If the 'mirror setoptions -enforcedefaultuser' option is set on the MAS and this parameter is specified, an error will be thrown. No companion password parameter is available for -RSuser. If this parameter is specified, the user will be prompted for the corresponding password. Therefore, specifying this parameter makes the 'mirror create' command an interactive command.

-script

-script <TCL_script>	The -script option allows you to specify the TCL script that defines the scripted mirror. The script must be in the syncinc/share/tcl, or custom/site/share/tcl directory. For information about how to create the TCL script, see the ENOVIA Synchroncity DesignSync Administrator's Guide. The -script option is mutually exclusive with the -type option.

-selector

-selector <list>	The -selector option is used to choose which versions of the files in the vault to place in the mirror directory. This value's syntax is checked to make sure the selector is valid. The default is 'Trunk:Latest'.

-type (Module-based)

-type	The type of the mirror. The only type supported for modules is 'normal.' This option is mutually exclusive with the -script option.

-type (File-based)

```
-type           The type of the mirror. This must be 'normal',
                'primary', or 'secondary'. Secondary mirrors require
                the -primaryserver parameter and may specify the
                -PMASuser parameter. The default is
               'normal'.

                This option is mutually exclusive with the -script
                option.
```

-vaultURL

```
-vaultURL       Specifies the URL of the vault directory whose
                contents will be mirrored out. This parameter is
               required. Specify the URL as follows:
                sync://<host>[:<port>]/path_to_vault or
                syncs://<host>[:<port>]/path_to_vault where
                'sync://' or 'syncs://' are required,
                <host> is the remote server,
                <port> is the remote server's port number
                (defaults to 2647)
                and the path_to_vault is the path from the remote
                servers root to the vault directory being mirrored.
```

RETURN VALUE

```
Returns an empty string on success.
```

SEE ALSO

```
mirror delete, mirror disable, mirror edit,
mirror enable, mirror get, mirror getoptions, mirror isenabled,
mirror ismirror, mirror list, mirror rename, mirror reset,
mirror setoptions, mirror status, mirrorsetdefaultuser
```

EXAMPLES

```
This example creates a primary mirror on the MAS sync://faure:30138.
The default user, as specified in the 'mirror setoptions' example, is
used for mirror communications. The mirror is not initially enabled.

stcl> mirror create -recursive -type primary -description "FCS builds" \
stcl> -noenable sync://faure:30138 -name releases -mirrordir \
stcl> /home/tbarbg2/Mirrors/releases -vaultURL \
stcl> sync://srv2.ABCo.com:2647/releases
stcl>
```

mirror delete

mirror delete Command

NAME

```
mirror delete       - Deletes a mirror
```

DESCRIPTION

```
This command deletes the mirror definition from both the MAS and the
RS. It does not stop any updates that are in progress on the mirror's
behalf or remove the mirror directory. If the MAS can be contacted
but the mirror cannot be removed because the RS cannot be contacted,
the mirror will be placed in a disabled state.

Generated mirrors cannot be removed. Scripted mirrors, which are used
to generate and modify the generated mirrors can be deleted.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.
```

SYNOPSIS

```
mirror delete [-force] -name <name> <serverURL>
```

ARGUMENTS

- Server URL

Server URL

```
serverURL       Specifies the URL of the MAS where the mirror will be
                deleted. Specify the URL as follows:
                sync://<host>[:<port>] or
                syncs://<host>[:<port>] where 'sync://' or 'syncs://'
                are required, <host> is the machine on which the
                SyncServer is installed, and <port> is the SyncServer
                port number (defaults to 2647/2679).
                For example: sync://serv1.abco.com:1024
```

OPTIONS

- -force
- -name

-force

```
-force          Specifying this option deletes the mirror definition
                from the MAS (to clean up the status output and stop
                unwanted email) when the RS is not available.

                Note: Use this option only when the RS is not available
                and is not likely to become available.

                When the -force option is used and the RS is not
                accessible, the user gets the following message:
                "Connect failure. Server '<host>:<port>' may have
                reset the connection."  But the mirror definition is
                removed from the MAS.
```

-name

```
-name           Name of the mirror to delete.

                Note: Generated mirrors cannot be removed.  Generated
                mirrors use the <TCL_script>@<script_assigned_name>.
                Any mirror containing a "@" is a generated mirror.

                The -name option is required.
```

RETURN VALUE

```
Returns an empty string on success.
```

SEE ALSO

```
mirror create, mirror disable, mirror edit,
mirror enable, mirror get, mirror getoptions, mirror isenabled,
mirror ismirror, mirror list, mirror rename, mirror reset,
mirror setoptions, mirror status, mirrorsetdefaultuser
```

EXAMPLES

Example Showing Deleting a Mirror

```
This example deletes the 'Releases' mirror from the MAS
```

```
sync://giovannelli:30138.

  stcl> mirror delete sync://giovannelli:30138 -name Releases
  stcl>
```

Example Showing That the Mirror Cannot Be Deleted

```
This example cannot delete the 'NML8B0' mirror from the MAS
sync://qechrhl02:30046 as the RS is disabled.

  stclc> mirror delete sync://qechrhl02:30046 -name NML8B0
  Connect failure. Server 'sting:30046' may have reset the connection.

  Could not remove the mirror's definition from the repository server.
  The mirror is disabled: NML8B0
  - Attempting to contact repository server...
  - som-E-11: Communication Connect Failure.
```

Example Using The Force Option to Delete a Mirror

```
 This example deletes the 'NML8B0' mirror from the MAS
 sync://qechrhl02:30046 even though the RS is disabled using the
 -force option.

  stclc> mirror delete sync://qechrhl02:30046 -force -name NML8B0
  Connect failure. Server 'sting:30046' may have reset the connection.

  stcl> mirror ismirror sync://qechrhl02:30046 -name NML8B0
  0
```

mirror disable

mirror disable Command

NAME

```
mirror disable      - Disables a mirror
```

DESCRIPTION

```
The mirror disable command disables a single mirror or all mirrors
depending on the parameter specified. When a mirror is disabled, it
is marked disabled on the MAS and an attempt is made to remove the
mirror definition from the mirror's repository server. Failing to
remove the definition from the repository server does not cause an
error. The RS mirror definition is replaced when the mirror is
```

```
enabled or removed when the mirror is deleted.

If an error occurs while processing one mirror in a list (during -all
switch) all remaining mirrors are processed. An error is thrown if
all mirrors being processed fail. If only one mirror is being
processed, an empty string is returned on success. If multiple
mirrors are processed, an error status message is printed for each
mirror that fails and the return value shows success and failure
status in the form { succeeded 3 failed 2 }.

Note: When a parent scripted mirror is disabled, the mirrors
generated by that script are also disabled. Additionally, the script
can disable a generated mirror by returning a status value of 2,
followed by a list of the generated mirrors to disable in the Mirrors
list.

Generated mirrors use the format <MirrorName>@<script_assigned_name>.
Any mirror containing a "@" is a generated mirror.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.
```

SYNOPSIS

```
mirror disable <serverURL> -all | -name <name>
```

ARGUMENTS

- Server URL

Server URL

```
serverURL       Specifies the URL of the MAS where the mirror will be
                disabled. Specify the URL as follows:
                sync://<host>[:<port>] or
                syncs://<host>[:<port>] where 'sync://' or 'syncs://'
                are required, <host> is the machine on which the
                SyncServer is installed, and <port> is the SyncServer
                port number (defaults to 2647/2679).
                For example: sync://serv1.abco.com:1024
```

OPTIONS

- -all
- -name

-all

```
-all            Evaluates all mirrors on the server.
```

-name

```
-name           Name of the mirror to disable.
```

RETURN VALUE

```
If only one mirror is being processed, an empty string is returned on
success. If multiple mirrors are being processed, an error status
message is printed for each mirror that fails and the return
value shows success and failure status in the form
{ succeeded 3 failed 2 }.
An error is thrown if all of the mirrors being processed fail.
```

SEE ALSO

```
mirror create, mirror delete, mirror edit,
mirror enable, mirror get, mirror getoptions, mirror isenabled,
mirror ismirror, mirror list, mirror rename, mirror reset,
mirror setoptions, mirror status, mirrorsetdefaultuser
```

EXAMPLES

```
This example disables the "Releases" mirror on the MAS
sync://giovannelli:30138.

stcl> mirror disable sync://giovannelli:30138 -name Releases
stcl>
```

mirror edit

mirror edit Command

NAME

```
mirror edit         - Modifies mirror parameters
```

DESCRIPTION

- Notes for File-Based and Legacy Module Objects

```
The mirror edit command modifies specified mirror parameters
```

```
Passwords are never specified on the command line for this command.
When passwords are needed, you are prompted and the command
becomes interactive.

Note: You cannot change the name of the mirror using mirror edit.  To
change the mirror name, use the 'mirror rename' command.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.
```

Notes for File-Based and Legacy Module Objects

```
Vaults containing DesignSync references always recursively mirrors
the contents of their reference vaults.
```

SYNOPSIS

```
mirror edit [-cachedir <path>] [-cachelinktype hard|soft]
            [-category <category>] [-description <description>]
            [-fetchstate get|share] [-hrefmode <static|dynamic|normal>]
            [-MASuser <user>] [-mcachemode link | server]
            [-mirrordir <mirrorDir>] -name <name> [-notify <email_list>]
            [-PMASuser <user>] [-primaryserver <serverURL>]
            [-[no]recursive] [-RSuser <user>] [-vaultURL <vaultURL>]
            [-selector <selector_list>] [-script <TCL_script>]
            [-type normal|primary|secondary] <serverURL>
```

ARGUMENTS

- Server URL

Server URL

```
serverURL       Specifies the URL of the MAS where the mirror will be
                edited. Specify the URL as follows:
                sync://<host>[:<port>] or
                syncs://<host>[:<port>] where 'sync://' or 'syncs://'
                are required, <host> is the machine on which the
                SyncServer is installed, and <port> is the SyncServer
                port number (defaults to 2647/2679).
                Example: sync://serv1.abco.com:1024
```

OPTIONS

- -cachedir
- -cachelinktype

-cachedir

```
-cachedir <path>    The -cache option allows you specify the path to the
                    mirror specific file cache. If you update mirrors
                    with the share state and no file cache is specified,
                    the default cache or project caches are used. The
                    default cache is determined by the MUP's registry
                    files which include the server's registry files and
                    the MirrorRegistry.reg file.
```

-cachelinktype

```
-cachelinktype      The -cachelinktype option indicates whether creating
 hard|soft          a mirror with cache links uses hard or soft
                    (symbolic) links in the mirror. This option is
                    ignored if the -fetchstate get option is used.

                    "-cachelinktype hard" populates the mirror with hard
                    links to the cache.  Hard links require that mirror
                    and the cache be on the same disk partition. If the
                    system cannot create hard links to the cache, it
                    will switch automatically to creating soft links to
                    the cache.

                    "-cachelinktype soft" populates the mirror with
                    symbolic links to the cache. (Default)

                    Note: This option does not use the registry settings
```

that determine whether a hard link or soft link should be used. The default setting for this option is to use hard links.

-category

-category — A parameter to assign a category (arbitrary string) to a mirror. All mirrors belonging to a category can be selected by using the -category parameter of 'mirror list'. A category must be composed of the following set [A-Za-z0-9_/-.] or a space.

-description

-description — An optional description of the mirror. A description must be composed of the following set [A-Za-z0-9_/-.] or a space.

-fetchstate

-fetchstate get|share — An optional mode to indicate whether the mirror should be populated with local copies, or populated with file cache links.

"-fetchstate get" populates the mirror with local copies. (Default)

"-fetchstate share" populates the mirror with links the cache. You can use the -cachelinktype to specify whether the links should be soft links (symbolic links) or hard links.

-hrefmode (Module-based)

-hrefmode — An optional mode to enable the selection of a hierarchical reference mode when recursively populating the mirror for a module. The default mode is "normal". The other modes are "static" and "dynamic". This option is only meaningful when <vaultURL> refers to a non-legacy module.

-MASuser

```
-MASuser        The name of the user to establish communication from
                the repository server (RS) to the mirror administration
                server (MAS). If the 'mirror setoptions
                -enforcedefaultuser' option is set on the MAS and this
                parameter is specified, an error will be thrown. If
                this parameter is specified, the user will be prompted
                for the corresponding password. Therefore, specifying
                this parameter makes the 'mirror edit' command an
                interactive command.
```

-mcachemode (Module-based)

```
-mcachemode
 link|server     When using the scripted mirror capability to populate
                 an auto-generated mirror, this option specifies:
                 o link - Attempt to create mcache links to referenced
                 submodules by searching the mirror directory supplied
                 with the scripted/autogen mirror.
                 o server - Fetch the submodules from the server.
                 When populating a normal mirror, this option
                 specifies:
                 o link - Attempt to create mcache links to
                 referenced submodules by searching using the
                 default mcache paths defined in the registry.
                 o server - Fetch the submodules from the server.
```

-mirrordir (Module-based)

```
-mirrordir      Pathname of the mirror directory. This pathname must
                be unique with respect to all other mirrors defined on
                the mirror administration server (MAS).

                If this pathname contains a relative path with
                respect to the stcl's cwd, it will be converted to an
                absolute pathname BEFORE the name is passed to the
                mirror administration server.

                If the mirror directory pathname contains a symbolic
                link (UNIX only), the user controls whether the
                symbolic link is resolved or used as is, with the
                EnableRealMirrorPaths registry key.

                The '~' home directory character is allowed (UNIX
                only). The directory pathname must be relevant on the
                same LAN as the mirror administration server. The
                directory name can be prefixed with 'file://' if an
                absolute pathname is specified.

               The path of a mirror directory can be shared by other
                mirrors defined on the MAS or used for multiple
                modules. Since the base directory for multiple
```

modules can be the same, the base directory can work as a mirror directory.

You cannot use a mirror directory for a module mirror that is already used for a legacy module or a DesignSync vault.

-mirrordir (File-based)

-mirrordir — Pathname of the mirror directory. This pathname must be unique with respect to all other mirrors defined on the mirror administration server (MAS).

If this pathname contains a relative path with respect to the stcl's cwd, it will be converted to an absolute pathname BEFORE the name is passed to the mirror administration server.

If the mirror directory pathname contains a symbolic link (UNIX only), the user controls whether the symbolic link is resolved or used as is, with the EnableRealMirrorPaths registry key.

The '~' home directory character is allowed (UNIX only). The directory pathname must be relevant on the same LAN as the mirror administration server. The directory name can be prefixed with 'file://' if an absolute pathname is specified.

The mirror directory path and the relative path of the legacy module must be unique. You cannot use a directory for a legacy module that is already in use by another mirror.

-name

-name — Name of the mirror to edit. This parameter is required.

-notify

-notify — A comma-separated or space-separated list of email addresses and/or user names to send email to whenever the mirror generates notifications. The defaultnotifylist (see 'mirror setoptions') will be appended to this list. If user names are specified, the users must have user profiles on the MAS servers.

-PMASuser

```
-PMASuser       The name of the user to establish communication from
                the mirror administration server (MAS) to the primary
                mirror administration server (PMAS). If the mirror's
                type is not set to "secondary", this parameter is
                silently ignored. If the 'mirror setoptions
                -enforcedefaultuser' option is set on the MAS and
                this parameter is specified, an error will be thrown.
                If this parameter is specified, the user will be
                prompted for the corresponding password. Therefore,
                specifying this parameter makes the 'mirror edit'
                command an interactive command.
```

-primaryserver

```
-primaryserver   Specifies the URL of the SyncServer hosting the
                 primary mirror (PMAS).  If the mirror's type is not
                 set to "secondary", this parameter is silently
                 ignored.  Specify the URL as follows:
                 sync://<host>[:<port>] or syncs://<host>[:<port>]
                 where 'sync://' or 'syncs://' are required, <host>
                 is the machine on which the PMAS is installed, and
                 <port> is the PMAS port number (defaults to 2647.
                 Example: -primaryserver
                 sync://serv1.abco.com:1024
```

-[no]recursive (Module-based)

```
-[no]recursive  Determines whether to mirror the module's contents
                only or the entire module hierarchy.

                -recursive specifies that the mirror populates a
                module's contents and the contents of all its
                sub-modules recursively. (Default)

                -norecursive specifies that the mirror
                populates a module without processing the module
                hierarchy. This does not remove the sub-module
                mirrors previously created.
```

-RSuser

```
-RSuser         The name of the user to establish communication from
                the mirror administration server (MAS) to the
                repository server (RS). This User must have a user
                profile on the RS. If the 'mirror setoptions
```

	-enforcedefaultuser' option is set on the MAS and this parameter is specified, an error will be thrown. No companion password parameter is available for -RSuser. If this parameter is specified, the user is prompted for the corresponding password. Therefore, specifying this parameter makes the 'mirror edit' command interactive.

-script

-script <TCL_script>	The -script option allows you to specify the TCL script that defines the scripted mirror. The script must be in the syncinc/share/tcl, or custom/site/share/tcl directory. For information about how to create the TCL script, see the ENOVIA Synchroncity DesignSync Administrator's Guide. IMPORTANT: You cannot use the -script option to change the mirror type. You can only use it to change which script controls the scripted mirror. The -script option is mutually exclusive with the -type option.

-selector

-selector	The selector_list to use to choose which versions of the files in the vault to place in the mirror directory. This value's syntax will be checked to make sure the selector is valid. With the exception of a scripted mirror, you cannot edit the selector when the mirror reflects a module.

-type (Module-based)

-type	The mirror type is always 'normal.'

-type (File-based)

-type	Specify the type of the mirror as 'normal', 'primary', or 'secondary'. Secondary mirrors require the -primaryserver parameter and may specify the -PMASuser parameter.

-vaultURL

```
-vaultURL      Specifies the URL of the vault directory whose
               contents will be mirrored out. Specify the URL as
              follows: sync://<host>[:<port>]/path_to_vault or
               syncs://<host>[:<port>]/path_to_vault where
               'sync://' or 'syncs://' are required, <host> is the
               remote server,  <port> is the remote server's port
               number (defaults to 2647), and the path_to_vault is
               the path from the remote servers root to the vault
               directory to be mirrored out.

               Note:
               o You can edit the vault URL from a DS folder to a
                 legacy module and vice versa.
               o You can edit the vault URL from one module folder to
                 another module folder.
               o You cannot edit the vault URL when the mirror
                 reflects a module.
               o You cannot edit the vault URL from a legacy module
                 or a DS vault to non-legacy module or vice
                 versa. If a mirror is created to populate a legacy
                 module, it cannot be modified to populate a
                 non-legacy module. Likewise, if a mirror is created
                 to populate a non-legacy module, it cannot be
                 modified to populate a legacy module. An error is
                 generated if such an attempt is made.
```

RETURN VALUE

```
Returns an empty string on success.
```

SEE ALSO

```
mirror create, mirror delete, mirror disable,
mirror enable, mirror get, mirror getoptions, mirror isenabled,
mirror ismirror, mirror list, mirror rename, mirror reset,
mirror setoptions, mirror status, mirrorsetdefaultuser
```

EXAMPLES

```
This example changes the mirror directory for the "releases" mirror that
was created in the 'mirror create' example. The 'mirror rename' example
shows how to change the mirror name.

stcl> mirror edit sync://faure:30138 -name releases -mirrordir \
stcl> /home/tbarbg2/Mirrors/Releases
stcl>
```

mirror enable

mirror enable Command

NAME

```
mirror enable       - Enables a specified mirror
```

DESCRIPTION

```
This command enables a single mirror, all disabled mirrors, or all
mirrors depending on the parameter specified. When a mirror is enabled,
it is effectively re-registered. The repository server for each mirror
will be contacted and the mirrors definition on the repository server
replaced. If the mirror cannot re-register, the mirror will remain in
(or be moved to) the disabled state.

Note that enabling an already  enabled mirror could result in the
mirror becoming disabled if the RS  is unreachable. If an error occurs
while processing one mirror in a  list (-disabled or -all switches) all
remaining mirrors will be  processed. An error is thrown if all mirrors
fail.

Note: When a parent scripted mirror is enabled, the mirrors generated
by that script are also enabled. Additionally, if the script returns
a list of mirrors to generate, then those mirrors are automatically
enabled, if they were in the disabled state.
Generated mirrors use the format:
<MirrorName>@<script_assigned_name>.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.
```

SYNOPSIS

```
mirror enable <serverURL> -all | -disabled | -name name
```

ARGUMENTS

- Server URL

Server URL

```
serverURL       Specifies the URL of the MAS where the mirror will be
                enabled. Specify the URL as follows:
                sync://<host>[:<port>] or
                syncs://<host>[:<port>] where 'sync://' or 'syncs://'
```

```
are required, <host> is the machine on which the
SyncServer is installed, and <port> is the SyncServer
port number (defaults to 2647/2679).
Example: sync://serv1.abco.com:1024
```

OPTIONS

- -all
- -disabled
- -name

-all

```
-all            Enables all the mirrors on the server.

                This option is mutually exclusive with -disabled and
                -name.
```

-disabled

```
-disabled       Enables all all mirrors on the server
                that are currently in the disabled state.

                This option is mutually exclusive with -all and
                -name.
```

-name

```
-name <name>    Name of the mirror to enable.
                Note: Generated mirrors cannot be enabled or
                disabled.  If the parent scripted mirror is enabled,
                that, in turn, enables any disabled generated mirrors
                generated by that scripted mirror. Generated mirrors
                use the name <MirrorName>@<script_assigned_name>. Any
                mirror containing a "@" is a generated mirror.

                This option is mutually exclusive with -all and
               -disabled.
```

RETURN VALUE

```
If only one mirror is being processed, an empty string is returned on
success. If multiple mirrors are being processed, an error status
message is printed for each mirror that fails and the return
value shows success and failure status in the form
{ succeeded 3 failed 2 }. An error is thrown if all of the mirrors
being processed fail.
```

SEE ALSO

```
mirror create, mirror delete, mirror disable, mirror edit,
mirror get, mirror getoptions, mirror isenabled,
mirror ismirror, mirror list, mirror rename, mirror reset,
mirror setoptions, mirror status, mirrorsetdefaultuser
```

EXAMPLES

```
This example enables the "Releases" mirror.

stcl> mirror enable sync://faure:30138 -name Releases
stcl>
```

mirror get

mirror get Command

NAME

```
mirror get          - Returns all parameters for specified mirror
```

DESCRIPTION

```
The mirror get command returns all the parameters for a specified
mirror. The following parameters are set to an empty
string if they are not relevant or cannot be derived:

  'primaryserver'
  'usingdefPMASuser',
  'PMASuser'

The following parameters return 1 or 0 when the format is 'list'
and 'True' or 'False' when the format is 'text':

 'modulerecursion',
 'usingdefaultRSuser'
 'usingdefaultMASuser'
 'usingdefaultPMASuser'
 'enabled'

The defaultuser is the user specified with:

 'mirror setoptions -defaultuser'

Note: When a generated mirror is specified, the parameters returned
are for the scripted mirror definitions with only the following
```

```
parameters coming directly from the generated mirror:
'vault'
'selector'
'mirror directory'
'script' (empty)
'type' (normal)

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.
```

SYNOPSIS

```
mirror get <serverURL> [-format text|list] -name <name>
```

ARGUMENTS

Server URL

```
serverURL       Specifies the URL of the MAS to get the mirror
                parameters from. Specify the URL as follows:
                sync://<host>[:<port>] or
                syncs://<host>[:<port>] where 'sync://' or 'syncs://'
                are required, <host> is the machine on which the
                SyncServer is installed, and <port> is the
                SyncServer port number (defaults to 2647/2679).
                Example:  sync://serv1.abco.com:1024
```

OPTIONS

-format

```
-format         Specifies the way the output is returned.
                The default is text. The format 'text' will return
                each mirror parameter on a new line in the format
                name=value. The format 'list' will list the values in
                a Tcl list in the form {name1 value1 name2 value2 ...}
```

-name

```
-name <name>    Name of the mirror.  This parameter is required.
```

RETURN VALUE

```
Returns the parameters for the mirror. The following parameters are
returned:
name
mirrordir
vaultURL
selector
script
category
description
notifylist
commonnotifylist
modulerecursion
hrefmode
type
enabled
primaryserver
defaultuser
RSuser
MASuser
PMASuser
usingdefaultRSuser
usingdefaultPMASuser
usingdefaultMASuser
fetchstate
cachelinktype
cachedir
mcachemode
```

SEE ALSO

```
mirror create, mirror delete, mirror disable, mirror edit,
mirror enable, mirror getoptions, mirror isenabled,
mirror ismirror, mirror list, mirror rename, mirror reset,
mirror setoptions, mirror status, mirrorsetdefaultuser
```

EXAMPLES

Example Showing the Parameters for a Non-Scripted Mirror

```
This example returns the parameters for the "Releases" mirror
(non-scripted) on the MAS sync://faure:30138.

stcl> mirror get sync://faure:30138 -name Releases
```

```
 name                  = Releases
 mirrordir             = /home/barbg/Mirrors/Releases
 vaultURL              = sync://srv2.ABCo.com:2647/releases
 selector              = Trunk:Latest
 script                =
 category              =
 description           = FCS builds
 type                  = primary
 enabled               = True
 modulerecursion       = True
 hrefmode              = Normal
 primaryserver         =
 notifylist            =
 commonnotifylist      = barbg
 defaultuser           = barbg
 usingdefaultMASuser   = True
 MASuser               = barbg
 usingdefaultRSuser    = True
 RSuser                = barbg
 usingdefaultPMASuser = False
 PMASuser              =
 fetchstate            = get
 cachelinktype         = soft
 cachedir              =
 mcachemode            = server
 stcl>
```

Example Showing the Parameters for a Scripted Mirror

```
 This example returns the parameters for the "AUTO_RELEASES" scripted
 mirror on the MAS sync://qewflx10:30047.

stcl> mirror get sync://qewflx10:30047 -name SCR_1
 name                  = AUTO_RELEASES
 mirrordir             = /home/tlarry1/Modules/mirrors/DS_AUTO_RELEASES
 vaultURL              = sync://qewflx10:30047/Modules/Blocks
 selector              = REL_*
 script                = generate_mirror.tcl
 category              =
 description           =
 type                  = autogen
 enabled               = True
 modulerecursion       = True
 hrefmode              = normal
 primaryserver         =
 notifylist            =
 commonnotifylist      = masteradmin
 defaultuser           = masteradmin
 usingdefaultMASuser   = True
 MASuser               = masteradmin
 usingdefaultRSuser    = True
 RSuser                = masteradmin
 usingdefaultPMASuser = False
 PMASuser              =
```

```
fetchstate           = share
cachelinktype        = hard
cachedir             = /home/tlarry1/Modules/sync_cache
mcachemode           = link
stcl>
```

mirror getoptions

mirror getoptions Command

NAME

```
mirror getoptions   - Gets mirror options on a server
```

DESCRIPTION

```
This command is used to get general mirror options for all mirrors on
a MAS or RS. Options that have not been set will return an empty
string. The following parameters will return 1 or 0 when the format is
'list' and 'True' or 'False' when the format is 'text':

  isdefaultuserenforced
  isRSenabled
  isMASenabled

The parameter warnifstale will return 'No' if the value is 0 and the
format is text.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.
```

SYNOPSIS

```
mirror getoptions <serverURL> [-format text|list]
```

ARGUMENTS

- Server URL

Server URL

```
serverURL       Specifies the URL of the SyncServer (RS or MAS).
                Specify the URL as follows:
                sync://<host>[:<port>] or
                syncs://<host>[:<port>] where 'sync://' or 'syncs://'
                are required,  <host> is the machine on which the
                SyncServer is installed, and <port> is the SyncServer
```

```
                port number (defaults to 2647/2679).
                Example: sync://serv1.abco.com:1024
```

OPTIONS

- -format

-format

```
-format         Specifies the way the output is returned. The
                default is text. The format 'text' will return each
                option on a new line in the format name=value.
                The format 'list' will list the options in a Tcl list
                in the form {name1 value1 name2 value2 ...}
```

RETURN VALUE

```
Returns the general options for the mirrors on a server.
The following parameters will be returned:
defaultuser
commonnotifylist
name
isdefaultuserenforced
isRSenabled
isMASenabled
warnifstale
isSUIDenforced
```

SEE ALSO

```
mirror create, mirror delete, mirror disable, mirror edit,
mirror enable, mirror get, mirror isenabled,
mirror ismirror, mirror list, mirror rename, mirror reset,
mirror setoptions, mirror status, mirrorsetdefaultuser
```

EXAMPLES

- Example Showing the Mirror Options
- Example Showing the Mirror Options in a Formatted List

Example Showing the Mirror Options

```
This example gets the general mirror options for the SyncServer
sync://faure:30138.

stcl> mirror getoptions sync://faure:30138
isRSenabled           = False
```

```
isMASenabled          = True
name                  = faure
defaultuser           = barbg
isdefaultuserenforced = True
isSUIDenforced        = False
commonnotifylist      = barbg
warnifstale           = No
isSUIDenforced        = Yes

stcl>
```

Example Showing the Mirror Options in a Formatted List

```
This example shows the "-format list" output format:

stcl> mirror getoptions sync://faure:30138 -format list
isRSenabled 0 isMASenabled 1 name faure:30138 defaultuser barbg
isdefaultuserenforced 1 isSUIDenforced 0 commonnotifylist barbg
warnifstale 0 isSUIDenforced 1
stcl>
```

mirror isenabled

mirror isenabled Command

NAME

```
mirror isenabled    - Tests whether a mirror is enabled
```

DESCRIPTION

```
Test if a mirror is enabled. Used primarily for scripting.

Note: Generated mirrors cannot be disabled directly as an
argument to this command. The mirror status command can be
used to determine if a generated mirror is enabled or disabled.
Generated mirrors are created with the name:
<MirrorName>@<script_assigned_name>.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.
```

SYNOPSIS

```
mirror isenabled <serverURL> -name <name>
```

ARGUMENTS

- Server URL

Server URL

```
serverURL       Specifies the URL of the MAS where the mirror is
                defined. Specify the URL as follows:
                sync://<host>[:<port>] or
                syncs://<host>[:<port>] where 'sync://' or 'syncs://'
                are required, <host> is the machine on which the
                SyncServer is installed, and <port> is the SyncServer
                port number (defaults to 2647/2679).
                Example: sync://serv1.abco.com:1024
```

OPTIONS

- -name

-name

```
-name           Name of the mirror.  This parameter is required.
```

RETURN VALUE

```
Returns 1 if the mirror is enabled or 0 if the mirror is disabled.
```

SEE ALSO

```
mirror create, mirror delete, mirror disable, mirror edit,
mirror enable, mirror get, mirror getoptions,
mirror ismirror, mirror list, mirror rename, mirror reset,
mirror setoptions, mirror status, mirrorsetdefaultuser
```

EXAMPLES

```
In this example, 'mirror isenabled' returns 1, because the "Releases"
mirror on the SyncServer sync://faure:30138 is enabled.

stcl> mirror isenabled sync://faure:30138 -name Releases
1
stcl>
```

mirror ismirror

mirror ismirror Command

NAME

```
mirror ismirror     - Tests whether a name is valid for a defined
                      mirror
```

DESCRIPTION

```
Test if a name is valid for an existing mirror. Used primarily for
scripting. Allows a name to be tested instead of one of the other
commands throwing an error if the mirror name was wrong.

Note: Generated mirrors cannot be validated using the mirror ismirror
command. Generated mirrors are created with the name
<MirrorName>@<script_assigned_name>.  Any mirror containing a "@" is
a generated mirror.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.
```

SYNOPSIS

```
mirror ismirror <serverURL> -name <name>
```

ARGUMENTS

- Server URL

Server URL

```
serverURL       Specifies the URL of the MAS where the mirror is
                defined. Specify the URL as follows:
                sync://<host>[:<port>] or
                syncs://<host>[:<port>] where 'sync://' or 'syncs://'
                are required, <host> is the machine on which the
                SyncServer is installed, and <port> is the SyncServer
                port number (defaults to 2647/2679).
                Example: sync://serv1.abco.com:1024
```

OPTIONS

- -name

-name

```
-name           Name of the mirror.
```

RETURN VALUE

```
Returns 1 if the mirror is defined on the server or 0 if the mirror
is not defined.
```

SEE ALSO

```
mirror create, mirror delete, mirror disable, mirror edit,
mirror enable, mirror get, mirror getoptions, mirror isenabled,
mirror list, mirror rename, mirror reset,
mirror setoptions, mirror status, mirrorsetdefaultuser
```

EXAMPLES

```
In this example, 'mirror ismirror' returns 1, because there is a "Releases"
mirror on the SyncServer sync://faure:30138.

stcl> mirror ismirror sync://faure:30138 -name Releases
1
stcl>
```

mirror list

mirror list Command

NAME

```
mirror list         - Returns a list of all mirrors from a server
```

DESCRIPTION

```
The mirror list command returns a list of all mirrors from a server
that matches a specified search criterion. Used primarily for
scripting. The -enabled, -disabled, and -all switches can be combined
with patterns from the -category and -name parameters to reduce the
number of mirrors returned in the list.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.
```

SYNOPSIS

```
mirror list <serverURL> [-category <category_pattern>]
```

```
            [-enabled|disabled|all] [-format text|list]
            [-name <name_pattern>]
```

ARGUMENTS

- Server URL

Server URL

serverURL	Specifies the URL of the MAS where the mirrors are defined. Specify the URL as follows: sync://<host>[:<port>] or syncs://<host>[:<port>] where 'sync://' or 'syncs://' are required, <host> is the machine on which the SyncServer is installed, and <port> is the SyncServer port number (defaults to 2647/2679). Example: sync://serv1.abco.com:1024

OPTIONS

- -all
- -category
- -disabled
- -enabled
- -format
- -name

-all

-all	Request a list of all mirrors from the server. This is the default.

-category

-category	A category pattern used to further reduce the list of mirrors returned. Mirror categories that do not match the pattern will be removed from the result list. A pattern is defined using UNIX glob-style notation. '*' matches any number of characters, '?' matches exactly one character, [chars] matches any characters in chars, and any other characters in the pattern are taken as literals that must match the input exactly.

-disabled

```
-disabled       Request a list of all disabled mirrors from the server.
```

-enabled

```
-enabled        Request a list of all enabled mirrors from the server.
```

-format

```
-format         Specifies the way the output is returned. The
                default is text.
                With the 'text' format, the matching mirrors are
                printed to the screen, one per line.
                With the 'list" format, the matching mirrors are
                returned as a Tcl list.
```

-name

```
-name           A name pattern used to further reduce the list of
                mirrors returned. Mirror names that do not match the
                pattern will be removed from the result list. A
                pattern is defined using UNIX glob-style notation.
                '*' matches any number of characters, '?' matches
                exactly one character,  [chars] matches any characters
                in chars, and any other characters in the pattern are
                taken as literals that must match the input exactly.
```

RETURN VALUE

```
Returns the names of all mirrors on the server that match the search
criteria. If no mirrors match, an empty string is returned.
```

SEE ALSO

```
mirror create, mirror delete, mirror disable, mirror edit,
mirror enable, mirror get, mirror getoptions, mirror isenabled,
mirror ismirror, mirror rename, mirror reset,
mirror setoptions, mirror status, mirrorsetdefaultuser
```

EXAMPLES

- Example Showing Enabled Mirrors in Text Format

Example Showing Enabled Mirrors in Text Format

```
This example returns all enabled mirrors on the SyncServer
sync://faure:30138. There are two mirrors M01 and M02 that
are enabled.

  stcl> mirror list sync://faure:30138 -enabled
  M01
  M02
```

Example Showing Enabled Mirrors in List Format

```
The output format is different if you use the -format list
option.

  stcl> mirror list sync://faure:30138 -enabled
       -format list
  M01 M02
```

mirror rename

mirror rename Command

NAME

```
mirror rename       - Changes the mirror name
```

DESCRIPTION

```
The mirror rename command changes the mirrors name from <name> to
<newname>. The repository server must be contacted for this command.

This command cannot be used to rename scripted or generated mirrors.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.
```

SYNOPSIS

```
mirror rename <serverURL> -name <name> -newname <newname>
```

ARGUMENTS

- Server URL

Server URL

serverURL	Specifies the URL of the MAS where the mirror is defined. Specify the URL as follows: sync://<host>[:<port>] or syncs://<host>[:<port>] where 'sync://' or 'syncs://' are required, <host> is the machine on which the SyncServer is installed, and <port> is the SyncServer port number (defaults to 2647/2679). Example: sync://serv1.abco.com:1024

OPTIONS

- -name
- -newname

-name

-name	Current name of the mirror.

-newname

-newname	New name for the mirror. Names must be composed of the following set [A-Za-z0-9_/-.]. Mirror names cannot begin with a dash (-).

RETURN VALUE

Returns an empty string on success.

SEE ALSO

mirror create, mirror delete, mirror disable, mirror edit, mirror enable, mirror get, mirror getoptions, mirror isenabled, mirror ismirror, mirror list, mirror reset, mirror setoptions, mirror status, mirrorsetdefaultuser

EXAMPLES

```
This example changes the name of the "releases" mirror, created in the
'mirror create' example, to "Releases".

stcl> mirror rename sync://faure:30138 -name releases -newname Releases
stcl>
```

mirror requeue

mirror requeue Command

NAME

```
mirror requeue      - Requeue a transaction in the mirror
```

DESCRIPTION

```
This command provides a manual option to requeue mirror transactions
stored in the transaction record. When a mirror is generated, the
transactions that need to be processed are stored in a transaction
log and submitted in batches for processing.  If the mirror fails for
some reason, the transactions can be manually resubmitted for
processing either for a single mirror or for all the mirrors on the
MAS.

For more information on mirror requeuing, see the ENOVIA
Synchronicity DesignSync Data Manager Administratorâ€™s Guide.
```

SYNOPSIS

```
mirror requeue -name <mirror> <ServerURL>
```

ARGUMENTS

- Server URL

Server URL

```
serverURL       Specifies the URL of the MAS where the mirror is
                defined. Specify the URL as follows:
                sync://<host>[:<port>] or syncs://<host>[:<port>]
                where 'sync://' or 'syncs://' are required,
                <host> is the machine on which the SyncServer is
                installed, and <port> is the SyncServer port number
                (if omitted, defaults to 2647/2679).
                Example: sync://serv1.abco.com:1024
```

OPTIONS

- -name

-name

```
name <mirror>   Name of the mirror.
                To requeue transactions for all mirrors, specify
                "*".
```

RETURN VALUE

```
Returns an empty string on success. If there is a failure, DesignSync
reports an appropriate error message.
```

SEE ALSO

```
mirror create, mirror delete, mirror disable, mirror enable
```

EXAMPLES

- Example of Requeuing Transactions for a Single Mirror
- Example of Requeuing Transactions for All Mirrors

Example of Requeuing Transactions for a Single Mirror

```
This example shows transactions being requeued for a single mirror.

dss> mirror requeue -name chipMirror1
 sync://serv1.ABCo.com:2647/Modules/Chip/Chip-419
 dss>
```

Example of Requeuing Transactions for All Mirrors

```
This example shows transactions being requeued for all mirrors.

dss> mirror requeue -name *
 sync://serv1.ABCo.com:2647/Modules/Chip/Chip-419
 dss>
```

mirror reset

mirror reset Command

NAME

```
mirror reset        - Populates the mirror's directory, leaving it in
                      the same state as if the mirror had been removed
                      and then repopulated
```

DESCRIPTION

```
This command performs a full populate of the mirror and leaves
the mirror directory in the same state as if the mirror had been
removed and then repopulated. Resetting a mirror also resets its
submirrors. If the mirror's submirrors are disabled, they are
re-enabled by the reset operation.

Note: You cannot reset a generated or scripted mirror.  If a scripted
mirror is specified, it is silently ignored.  If a generated mirror is
specified, you will see an error stating that the mirror cannot be
reset.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.
```

SYNOPSIS

```
mirror reset <serverURL> -name <name>
```

ARGUMENTS

- Server URL

Server URL

```
serverURL       Specifies the URL of the MAS where the mirror is
                defined. Specify the URL as follows:
                sync://<host>[:<port>] or syncs://<host>[:<port>]
                where 'sync://' or 'syncs://' are required,
                <host> is the machine on which the SyncServer is
                installed, and <port> is the SyncServer port number
                (if omitted, defaults to 2647/2679).
                Example: sync://serv1.abco.com:1024
```

OPTIONS

- -name

-name

```
name            Name of the mirror.
```

RETURN VALUE

```
Returns an empty string on success.
```

SEE ALSO

```
mirror create, mirror delete, mirror disable, mirror edit,
mirror enable, mirror get, mirror getoptions, mirror isenabled,
mirror ismirror, mirror list, mirror rename,
mirror setoptions, mirror status, mirrorsetdefaultuser
```

EXAMPLES

```
This example resets the "Releases" mirror on the MAS sync://faure:30138.

stcl> mirror reset sync://faure:30138 -name Releases
stcl>
```

mirror setoptions

mirror setoptions Command

NAME

```
mirror setoptions   - Sets general mirror options
```

DESCRIPTION

```
The mirror setoptions command sets general mirror options for all
mirrors on a MAS or RS. Passwords are never specified on the
command line for this command. For a UNIX command-line version of
the -defaultUser option, see the mirrorsetdefaultuser shell script
command.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.
```

SYNOPSIS

```
mirror setoptions <serverURL> [-commonnotifylist<email_list>]
                  [-defaultuser <user>] [-disableMAS | -enableMAS]
                  [-disableRS | -enableRS] [-[no]enforcedefaultuser]
```

```
                [-[no]enforceSUID] [-name <name>]
                [-warnifstale <time>]
```

ARGUMENTS

- Server URL

Server URL

```
serverURL       Specifies the URL of the SyncServer (RS or MAS).
                Specify the URL as follows:
                sync://<host>[:<port>] or
                syncs://<host>[:<port>] where 'sync://' or 'syncs://'
                are required,  <host> is the machine on which the
                SyncServer is installed, and <port> is the SyncServer
                port number (defaults to 2647/2679).
                Example: sync://serv1.abco.com:1024
```

OPTIONS

- -commonnotifylist
- -defaultuser
- -disableMAS
- -disableRS
- -enableMAS
- -enableRS
- -enforcedefaultuser
- -enforceSUID
- -name
- -noenforcedefaultuser
- -noenforceSUID
- -warnifstale

-commonnotifylist

```
-commonnotifylist A comma-separated or space-separated list of email
                  addresses and/or user names. This list is internally
                  appended to the notify list specified with the
                  'mirror create' command for each mirror. The combined
                  list is used to send email whenever a mirror
                  generates notifications. The default is a list
                  of all Users who have the AdministrateServer AC
                  right. This option is relevant only for an MAS.
                  Notify lists are not defined on the RS but passed
                  to it during the enabling of a mirror.
```

-defaultuser

```
-defaultuser    The default user to be used when one of the needed
                users is not specified for the 'mirror create' command.
                If this parameter is specified, you are prompted for
                the default user's password. Specifying this parameter
                makes 'mirror setoptions' an interactive command.
                This option is only relevant for an MAS.
```

-disableMAS

```
-disableMAS     Disables a server as mirror administration server
```

-disableRS

```
-disableRS      Do not declare a server a repository server.
```

-enableMAS

```
-enableMAS      Enables a server a mirror administration server
```

-enableRS

```
-enableRS       Declare a server a repository server.
```

-enforcedefaultuser

```
-enforcedefaultuser   Turn on the default user policy.
                      When -enforcedefaultuser is in effect, users
                      cannot be specified during the 'mirror create'
                      command. This allows a company to establish a
                      user as the mirror administration user (default
                      user) for all servers and enforce that policy.
                      This option is only relevant for a MAS.
```

-enforceSUID

```
-enforceSUID    Enable enforcement of SUID for creating or moving
                mirrors. This option sets the UNIX permissions on
                new mirror directories to the default setting of 755.
                This option is relevant only for an MAS.
```

-name

```
-name           Declare a user-friendly name for a MAS server.
                If this option is not set, all references to the MAS
                in status reports will use its hostname:port
                identifier. No attempt will be made to assure this
                name is unique across all MAS servers in a corporation.
                This is the user's responsibility.
```

-noenforcedefaultuser

```
-noenforcedefaultuser Turn off the default user policy.
                      When -enforcedefaultuser is in effect, users
                      cannot be specified during the 'mirror create'
                      command. This allows a company to establish a
                      user as the mirror administration user (default
                      user) for all servers and enforce that policy.
                      This option is only relevant for a MAS.
```

-noenforceSUID

```
-noenforceSUID  Disable enforcement of SUID for creating or moving
                mirrors. If  this option is selected, permissions
                on new mirror directories are set to 777 and are
                open to any user.
```

-warnifstale

```
-warnifstale    Change a mirror's status to "Warning" if the mirror
                has not been up-to-date in the specified number of
                minutes. This parameter allows the user to identify
                mirrors that have had update processes running for a
                long period of time. This condition may be normal for
                the customer's data set but also may indicate a
                problem. A value of zero (the default) means the
                server never enters the warning status for this reason.
                Only values of 0, 10, 20, 30, 60, 120, 180, 360, 720,
                and 1440 are allowed.
```

RETURN VALUE

```
Returns an empty string on success.
```

SEE ALSO

```
mirror create, mirror delete, mirror disable, mirror edit,
mirror enable, mirror get, mirror getoptions, mirror isenabled,
mirror ismirror, mirror list, mirror rename, mirror reset,
mirror status, mirrorsetdefaultuser
```

EXAMPLES

```
This example defines the general mirror settings for the MAS
sync://faure:30138.

stcl> mirror setoptions sync://faure:30138 -defaultuser barbg \
stcl> -enforcedefaultuser -noenforceSUID -enableMAS -name faure \
stcl> -commonnotifylist barbg

Enter the password for the default user (barbg): ****

stcl>
```

mirror status

mirror status Command

NAME

```
mirror status       - Returns the status for the mirror
```

DESCRIPTION

- Understanding the Output

```
The mirror status command returns the status for the mirror
specified with <name> or all the mirrors on a server if the name is
not given. The <name> parameter is only valid when requesting status
of mirrors on a MAS. When status is requested from an RS, the RS will
make status calls to all the MASs that are registered with it. The
status of a mirror from the RS will be the same information as if the
status was requested directly from the MAS.
[This assumes the RS/MAS communication is working. If it is not,
status will only be shown for the mirrors where communication was
established with a message showing where it could not.]
The RS will include the MAS name where the mirror data came from in
the status.

Note: When you specify an RS as the -servertype, you cannot specify a
specific mirror name.
```

Understanding the Output

The output of the mirror status command can be formatted for easy viewing (-format text) or optimized for Tcl processing (-format list). Both viewing formats show the same information.

The mirror status command displays the following information for each mirror:

Property Names	Description
name	Unique name for the mirror defined when the mirror is created. The generated mirror name is in the format <MirrorName>@<Script_Assigned_Name>.
status	String indicating the health of the mirror. Possible values include: o good - There are no failures of any kind. o warning - A mirror update has failed and has not adequately been retried or the heartbeat is late. o failure - A mirror has repeatedly failed, the heartbeat was not received, or in the case of a scripted mirror, there was an error creating a generated mirror. o unknown - A mirror status is unknown. o incomplete - The mirror status file is incomplete or cannot be read. o disabled - the mirror is currently disabled.
lastuptodatetime	Last time the mirror was up-to-date. If the mirror is operating normally (no failures) then if there are no updates currently in progress for the mirror, this is the time of the last successful update or the last heartbeat, which ever is later. If there are updates in progress, this is the start time of the first update process to start.
mirrordir	The path to the mirror directory.
vaultURL	The Sync URL of the module, DesignSync vault or configuration for a legacy module being mirrored.
selector	The selector or tag applied to the objects in the mirror. Note: Even when a wildcard match was used to generate the mirror, you see the selector that matched the wildcard, not the wildcard value defined for the mirror.
category	User-defined category used to organize mirrors.
type	Type of mirror: o Normal - The default DesignSync mirror, which fetches design objects directly from the RS. Generated mirrors, even though they are

created from a scripted mirror are considered normal mirrors.

- Scripted - A mirror that has an associated Tcl script. The script determines which mirrors are automatically generated at a MAS based on the revision control operations occurring on the associated URL at the RS.

- Auto-generated (aka autogen) - Normal mirrors that get automatically generated by a scripted mirror. Autogen mirrors also go out of existence automatically when there's no revision control operations occurring on the associated URL at the RS.

- Primary - The primary mirror fetches design objects directly from the RS and serves them to secondary mirrors. This option is not supported for modules.

- Secondary - The secondary mirror fetches objects from a primary mirror instead of directly from the RS.

MASname	The name of the server hosting the mirror. This is the MAS's name as specified with 'mirror setoptions -name' or, if not set, the hostname:port for the MAS.
heartbeat	Time of last heartbeat received from the RS for the mirror.
lastupdatetime	The last time an update process successfully updated the mirror.
inprogress	A Boolean (0 or 1 for -format list and 'True' or 'False' for -format text) indicating if any update processes are running on the mirror's behalf.
firstfailuretime	(FFT)Time of first update process failure. This is 0 if there are no update failures. This time does not reflect heartbeat failures.
numberofretries	Number of update attempts to correct an update failure.
lastnotifytime	(LNT) The last time email was sent to all the recipients of the mirror's notify list.

This command is subject to access controls on the server. See the ENOVIA Synchronicity Access Control Guide for details.

SYNOPSIS

```
mirror status <serverURL> [-format text|list] [-name <name>]
              [-servertype RS|MAS]
```

ARGUMENTS

- Server URL

Server URL

```
serverURL       Specifies the URL of the MAS or RS to obtain mirror
                status from. Specify the URL as follows:
                sync://<host>[:<port>] or
                syncs://<host>[:<port>] where 'sync://' or 'syncs://'
                are required, <host> is the machine on which the
                SyncServer is installed, and <port> is the SyncServer
                port number (defaults to 2647/2679).
                Example: sync://serv1.abco.com:1024
```

OPTIONS

- -format
- -name
- -servertype

-format

```
-format         Specifies the way the output will be returned. The
 text|list      default is text. The format 'text' will output each
                mirror status parameter on a new line in the format
                name=value. When the status for more than one mirror
                is output, a blank line separates one mirror's
                status from another. The format 'list' will list the
                status in a Tcl list. If the <name> parameter is not
                specified, this command will return a list of lists
                with each sub-list holding the status of one mirror.
                The status for a mirror is in the form:
                {name1 value1 name2 value2 ...}.
```

-name

```
-name           Name of the mirror. This parameter is not allowed if
                the server type is 'RS' You can specified any mirror,
               including a generated or scripted mirror.
```

-servertype

-servertype RS|MAS	This parameter is set to either 'RS' for repository server or 'MAS' for mirror administration server. The default is 'MAS' if the server is acting as both a MAS and RS. Otherwise, this parameter defaults to the mode the server is configured for. This parameter specifies which mirror status the user is requesting. "mirror status" with "-servertype RS" may take some time to run. For each mirror that is mirroring data on the RS, the mirror's MAS must be contacted. For faster results, use the "mirror wheremirrored" command with the RS as the <vaultURL> argument. Use the "-status" option to "mirror wheremirrored" to report mirror status, and specify other "mirror wheremirrored" options to decrease the number of mirrors whose status is reported.

RETURN VALUE

Returns the status for a mirror or all the mirrors on a server. If only one mirror is being processed, the status is output (not returned) when the format is text or returned as a list when the format is list. If a mirror does not report its status, the command reports that as a failure in the form { succeeded 3 failed 2 }. An error is thrown if all mirrors being processed fail. If there are no mirrors defined, an error is thrown. Requesting status for a mirror server type that the server is not configured for is considered an error. If the format is list, the mirrors' status is returned in a list of lists as the last element in the return list. Example: { succeeded 4 failed 0 {{name ...} {name...} ...}}. The status for each mirror consists of a set of name value pairs.

SEE ALSO

mirror create, mirror delete, mirror disable, mirror edit, mirror enable, mirror get, mirror getoptions, mirror isenabled, mirror ismirror, mirror list, mirror rename, mirror reset, mirror setoptions, mirrorsetdefaultuser, mirror wheremirrored

EXAMPLES

- Example Showing Mirror Status for All Mirrors on an RS
- Example Showing Mirror Status for All Mirrors on an MAS
- Example Showing Mirror Status for the MAS in TCL List Format

Example Showing Mirror Status for All Mirrors on an RS

```
This example shows the status of all mirrors that are mirroring data on
the RS sync://srv2.ABCo.com:2647.

stcl> mirror status sync://srv2.ABCo.com:2647 -servertype RS
name             = Releases
status           = good
lastuptodatetime = 2005-12-29 05:37:14
mirrordir        = /home/tbarbg2/Mirrors/Releases
vaultURL         = sync://srv2.ABCo.com:2647/releases
selector         = Trunk:Latest
category         =
type             = primary
MASname          = faure
heartbeat        = 2005-12-29 05:36:12
lastupdatetime   = 2005-12-29 05:37:14
inprogress       = False
firstfailuretime = N/A
numberofretries  = 0
lastnotifytime   = N/A

name             = Releases
status           = good
lastuptodatetime = 2005-12-29 05:48:18
mirrordir        = /home/tbarbg7/Mirrors/Releases
vaultURL         = sync://srv2.ABCo.com:2647/releases
selector         = Trunk:Latest
category         =
type             = secondary
MASname          = giovannelli
heartbeat        = 2005-12-29 05:47:12
lastupdatetime   = 2005-12-29 05:48:18
inprogress       = False
firstfailuretime = N/A
numberofretries  = 0
lastnotifytime   = N/A


succeeded 2 failed 0
stcl>
```

Example Showing Mirror Status for All Mirrors on an MAS

```
This example shows the status of all mirrors on the MAS sync://src:2647
(2647 is the default cleartext port for DesignSync, so does not need to be
specified.)

stcl> mirror status sync://src -servertype MAS
name             = Trunky
status           = good
lastuptodatetime = 2006-04-11 11:38:23
mirrordir        = /home/syncmgr/mirrors/Trunky
vaultURL         =
sync://src.matrixone.net:2647/Projects/SyncInc/build_tools
selector         = Trunk:Latest
```

```
 category         = test
 type             = normal
 MASname          = devserver
 heartbeat        = 2006-04-11 11:38:23
 lastupdatetime   = 2006-04-11 06:58:27
 inprogress       = False
 firstfailuretime = N/A
 numberofretries  = 0
 lastnotifytime   = N/A

 name             = docs
 status           = good
 lastuptodatetime = 2006-04-11 11:38:23
 mirrordir        =
/home/syncmgr/sync_custom/servers/moniuszko/2647/htdocs/docs
 vaultURL         = sync://src.matrixone.net:2647/docs
 selector         = Trunk:Latest
 category         = devserver
 type             = normal
 MASname          = devserver
 heartbeat        = 2006-04-11 11:38:23
 lastupdatetime   = 2006-04-11 07:02:27
 inprogress       = False
 firstfailuretime = N/A
 numberofretries  = 0
 lastnotifytime   = N/A

 name             = testir
 status           = good
 lastuptodatetime = 2006-04-11 11:38:23
 mirrordir        =/home/syncmgr/sync_custom/servers/moniuszko
                   /2647/htdocs/testir
 vaultURL         = sync://src.matrixone.net:2647/Projects/testir
 selector         = Trunk:Latest
 category         = devserver
 type             = normal
 MASname          = devserver
 heartbeat        = 2006-04-11 11:38:23
 lastupdatetime   = 2006-04-11 06:54:06
 inprogress       = False
 firstfailuretime = N/A
 numberofretries  = 0
 lastnotifytime   = N/A


 succeeded 3 failed 0
 stcl>
```

Example Showing Mirror Status for the MAS in TCL List Format

```
 This example shows the "-format list" output format:

 stcl> mirror status sync://src -servertype MAS -format list
 succeeded 3 failed 0 {{name Trunky status good lastuptodatetime 1144769903
```

```
 mirrordir /home/syncmgr/mirrors/Trunky vaultURL
 sync://src.matrixone.net:2647/Projects/SyncInc/build_tools selector
Trunk:Latest
 category test type normal MASname devserver heartbeat 1144769903
lastupdatetime
 1144753107 inprogress 0 firstfailuretime 0 numberofretries 0 lastnotifytime
0}
 {name docs status good lastuptodatetime 1144769903 mirrordir
 /home/syncmgr/sync_custom/servers/moniuszko/2647/htdocs/docs vaultURL
 sync://src.matrixone.net:2647/docs selector Trunk:Latest category devserver
type
 normal MASname devserver heartbeat 1144769903 lastupdatetime 1144753347
 inprogress 0 firstfailuretime 0 numberofretries 0 lastnotifytime 0} {name
testir
 status good lastuptodatetime 1144769903 mirrordir
 /home/syncmgr/sync_custom/servers/moniuszko/2647/htdocs/testir vaultURL
 sync://src.matrixone.net:2647/Projects/testir selector Trunk:Latest category
 devserver type normal MASname devserver heartbeat 1144769903 lastupdatetime
 1144752846 inprogress 0 firstfailuretime 0 numberofretries 0 lastnotifytime
0}}
 stcl>
```

mirror wheremirrored

mirror wheremirrored Command

NAME

```
 mirror wheremirrored - Shows where vault data is mirrored
```

DESCRIPTION

- Pattern Matching
- Upgrading a Mirror with "wheremirrored" Information

```
 As an end user, use the "mirror wheremirrored" command to identify
 a mirror directory at your site to "setmirror" to. As a mirror
 administrator use the "mirror wheremirrored" command when defining a
 secondary mirror, to identify available primary mirrors. The "mirror
 wheremirrored" command can also be used to report the status of
 mirrors, and is faster than "mirror status" with the "-servertype RS"
 option. (When options to the "mirror wheremirrored" command are used
 to decrease the number of mirrors whose status is reported.) Only
 active (enabled) mirrors are reported by this command.
```

Pattern Matching

```
 Some of the options described below accept a pattern. A pattern is
 defined using UNIX glob-style notation (or the <pattern> described in
```

the Tcl "string match" command). "*" matches any number of characters, "?" matches exactly one character, and [chars] matches any characters in chars. Any other characters in the pattern are taken as literals that must match the input exactly. Also, \x an be used to match the single character x. This avoids the special interpretation of the characters *?[]\ in the pattern.

Upgrading a Mirror with "wheremirrored" Information

The information shown by the "mirror wheremirrored" command is for Repository Servers (RSs) that are running version V6R2008-9 of the software or higher. RSs should be upgraded to a version that supports "wheremirrored" first, so that the RS's are able to receive "wheremirrored" information sent to them by the Mirror Administration Servers (MASs). If a MAS is upgraded to a version with "wheremirrored" first, the "wheremirrored" upgrade step must be performed after each associated RS is upgraded.

If the information used by the "mirror wheremirrored" command is not present, the value "needs-upgrade" will be shown for most of the mirror properties. (The EXAMPLES section below has an example of this.) Similarly, the "wheremirrored" information shown for MASs at versions prior to the implementation of "wheremirrored" will have "needs-upgrade" for many of their mirror properties.

Once both the MAS and an RS are at a version with "wheremirrored" support, the mirror properties used by the "mirror wheremirrored" command are set ("upgraded") for an existing mirror, when one of these occur:
- The MAS is restarted, thereby restarting the mad (mirror administration daemon)
- The resetmirrordaemons command is used to restart the mad
- ProjectSync's "Reset MAS Daemon" is used
- A disabled mirror is enabled

The "upgrade" stores current information on the RS about a mirror, which the "wheremirrored" command uses. The "upgrade" occurs only once for each RS that has a mirror on the MAS.

Creating a mirror sets the properties used by the "mirror wheremirrored" command on the RS, for the mirror that was created. The mirror's "wheremirrored" properties are also updated when a mirror definition is edited.

This command is subject to access controls on the server. See the ENOVIA Synchronicity Access Control Guide for details.

SYNOPSIS

```
mirror wheremirrored <vaultURL> [-category <category>]
                     [-format text|list] [-name <name>]
```

```
                    [-selector <selector_list>]
                    [-status good|warning|failure|unknown|none|any]
                    [-type normal|primary|secondary|all]
```

ARGUMENTS

- Vault URL

Vault URL

```
<vaultURL>      Specifies the URL of the vault directory whose
                contents are being mirrored. This parameter is
                required. Specify the URL as follows:
                    sync[s]://<host>[:<port>][/path_to_vault]
                where the <host> and <port> are that of the RS.
                If the <port> is omitted, the default ports are
                used (2647 for cleartext and 2679 for SSL)

                The path_to_vault is the path on the RS that is
                being mirrored. The path_to_vault part of the
                <vaultURL> can be a pattern, defined using the UNIX
                glob-style notation described above. If a
                path_to_vault is not specified, all mirrors mirroring
                data anywhere on the RS will be reported. The options
                below can be used to filter the mirrors that are
                reported.
```

OPTIONS

- -category
- -format
- -name
- -selector
- -status
- -type

-category

```
-category       The <category> pattern used to filter the list of
                mirrors returned. Mirrors whose category does not
                match the <category> pattern will be removed from the
                result list. A pattern is defined using the UNIX
                glob-style notation described above. If a literal
                value is specified for the <category>, it must be an
                exact match of the category stored with the mirror
                definition. (Mirrors whose categories are not an exact
                match will be removed from the result list.)

                If a mirror definition has not been upgraded and this
```

option is specified, the mirror will not be in the result list. That's because the mirror does not yet have a category set in the "wheremirrored" properties on the RS, for the "mirror wheremirrored" command to use as a "-category" filter.

-format

-format — Specifies the way the output will be returned. The default is text. The format 'text' will output each mirror parameter on a new line in the format name=value. When the properties for more than one mirror is output, a blank line separates one mirror's properties from another. The format 'list' will list the properties in a Tcl list. The command will return a list of 5 elements with the last element being a list of lists, with each sub-list holding the properties of one mirror. The properties for a mirror will be in the form {name1 value1 name2 value2 ...}. The first 4 elements will be "succeeded number-succeeded failed number-failed". See the RETURN VALUE section below for more details.

-name

-name — The <name> pattern used to filter the list of mirrors returned. Mirror names that do not match the <name> pattern will be removed from the result list. A pattern is defined using the UNIX glob-style notation described above. If a literal value is specified for the <name>, it must be an exact match of the name stored with the mirror definition. (Mirrors whose names are not an exact match will be removed from the result list.)

-selector

-selector — The <selector_list> pattern used to filter the list of mirrors returned. Mirrors whose selector list does not match the <selector_list> pattern will be removed from the result list. The <selector_list> may be specified as a pattern, defined using the UNIX glob-style notation described above. If a literal value is specified for the <selector_list>, it must be an exact match of the selector stored with the mirror definition. (Mirrors whose selectors are not an exact match will be removed from the result list.)

However, branch selectors are normalized. I.e., a

<selector_list> value of "bugfix:" will match mirrors whose selectors are "bugfix:" or "bugfix:Latest". Similarly, the special branch value "Trunk" will match mirrors whose selectors are "Trunk:Latest" and "Trunk:".

-status

-status

The status of the mirror, used to filter the list of mirrors returned. The status specified must be either "good", "warning", "failure", "unknown", "none" or "any". Mirrors that do not match the status value will be removed from the result list.

The default value is "none", which means that status will not be reported. This is the most efficient value for the status, because the RS does not need to contact each MAS for every mirror that is reported. Mirror status is not reported, for "-status none".

Specifying a status value other than "none" will cause the RS to contact each MAS for every mirror that is reported. The status is the last criterion that is matched for a mirror. So, the request for status from a MAS is only sent by the RS if all other criteria are matched.

A status value of "any" will cause the RS to contact each MAS to get the status of every mirror (that matches the other criteria specified), but will not filter on the status value. The status value will be shown for all mirrors (that match the other criteria specified).

-type

-type

The type of the mirror, used to filter the list of mirrors returned. The type specified must be either "normal", "primary", "secondary" or "all". The default value is "all". Mirrors that do not match the type value will be removed from the result list.

If a mirror definition has not been upgraded and a type value other than "all" is specified, the mirror will not be in the result list. That's because the mirror does not yet have a type set in the "wheremirrored" properties on the RS, for the "mirror wheremirrored command to use as a "-type" filter.

RETURN VALUE

Returns properties and optionally status for each active (enabled) mirror meeting the specified criteria. When discussing the return value, the status of a mirror is considered to be a property.

When the default "-format text" is used, the properties are output and a succeeded/failed count is returned.

A specific mirror is only counted as a failure if there was a failure reporting the status of the mirror (when specifying a "-status" value other than "none"), and the mirror's status could not be retrieved from the MAS. If the mirror's status is "failure", then the "mirror wheremirrored" command has the mirror's "wheremirrored" return as succeeded, since the "mirror wheremirrored" command was successful in matching the mirror against the criteria specified with the command. Therefore, the "mirror wheremirrored" command counting a mirror as "succeeded" is independent of the status of the mirror. If a mirror is counted as "failed" due to not being able to get the status of a mirror, the "status" property for the mirror will have the value "unavailable".

When "-format list" is used, the properties and status for each mirror meeting the criteria is returned in a list of lists, as the last element in the return list. The first four elements of the returned list are name/value pairs with the number succeeded and number failed. This is similar to the return value of the "mirror status" command.

For example:
succeeded 4 failed 1 {{name val MASname val ...} {name val MASname val ...}}

An error will not be thrown if "-status" is used with a value other than "none", and no status could be retrieved for any of the matching mirrors (all mirrors reported as "failed" to get status).

An error is thrown if the command itself fails. Such as, if the server housing the vault is not enabled as an RS, if there are no active (enabled) mirrors registered with the RS, if the RS cannot be contacted, if multiple arguments are specified, or if the mirror name, category or selector contain illegal characters.

Mirrors created prior to their MAS being upgraded to a version that supports "where mirrored" do not have certain wheremirrored properties available at the RS. The value shown for those properties is "needs-upgrade". For details, see the "Upgrading a Mirror with wheremirrored Information" section above.

The following properties will be returned for each matching mirror:

name	The name of the mirror, which is only unique within a particular MAS.
MASname	The name of the server hosting the mirror, from the MAS's General Settings. This could have a value of "needs-upgrade".
MASurl	The URL of the MAS where the mirror is defined.

```
vaultURL        The URL of the vault that is being mirrored.
selector        The selector of the mirror.
mirrordir       The path to the mirror directory on the MAS's LAN.
                This could have a value of "needs-upgrade".
category        The category of the mirror on the MAS. This could
                have a value of "needs-upgrade".
type            The type of mirror: "normal", "primary", "secondary",
                or "needs-upgrade".
primaryServer   The MAS that is hosting the primary mirror. This
                could have a value of "needs-upgrade".

See the "mirror create" documentation for further details on the above
properties.

If the "-status" option is specified with a value other than the
default "none", then these additional properties are returned:

status          The mirror's status: "good", "warning", "failure",
                "unknown" or "unavailable". A status of "unavailable"
                is returned when the RS could not retrieve the status
                from the MAS. In which case, there may be a problem
                with the MAS, or the MAS may not be running.

                If a failure is encountered when trying to get the
                mirror status, the failure message will be reported
                when using the default "-format text" output. When
                "-format list" is used, the failure message will be
                returned in an additional "error" property.

error           The error message, if a failure is encountered when
                trying to get the mirror status. This property is
                only returned when "-format list" is used. If the
                default "-format text" is used, then this error
                message is output.
```

SEE ALSO

```
mirror create, mirror status, setmirror
```

EXAMPLES

Example Showing All Mirrors for a Server's Projects

```
To find all mirrors that are mirroring a server's projects:

stcl> mirror wheremirrored sync://qewflx10:30018/Projects/*
name             = Top
```

```
MASname          = North Carolina
MASurl           = sync://qewflx11.matrixone.net:30158
vaultURL         = sync://qewflx10:30018/Projects/Top
selector         = Trunk:Latest
mirrordir        = /home/tbarbg8/Mirrors/Secondary/Top/
category         = Test
type             = secondary
primaryserver    = sync://qewflx10:30148

name             = Test
MASname          = San Jose
MASurl           = sync://qewflx12.matrixone.net:30128
vaultURL         = sync://qewflx10:30018/Projects/Test
selector         = Trunk
mirrordir        = /home/tbarbg8/Mirrors/Normal/Test/
category         = Testing
type             = normal
primaryserver    =

...
```

Example Showing Failures in TCL List Format

```
The "-format list" return when a failure is encountered while getting
the status of a mirror:

stcl> mirror wheremirrored sync://qewflx10:30018 -name */ALU \
     -status any -format list
Connect failure. Server 'qewflx10.matrixone.net:30128' may have reset
                 the connection.

succeeded 2 failed 1 {{name Top/ALU MASname {San Jose} MASurl
sync://qewflx12.matrixone.net:30128 vaultURL
sync://qewflx10:30018/Projects/ALU selector Trunk:Latest mirrordir
/home/tbarbg8/Mirrors/Normal/Top/ALU/ category Production type normal
primaryserver {} status unavailable error {Failure getting status:
Attempting to contact mirror administration server...
- som-E-11: Communication Connect Failure.}} {name Top/ALU MASname
Cambridge MASurl sync://qewflx10.matrixone.net:30148 vaultURL
sync://qewflx10:30018/Projects/ALU selector Trunk:Latest mirrordir
/home/tbarbg8/Mirrors/Primary/Top/ALU/ category Development type
primary primaryserver {} status good} {name Top/ALU MASname
{North Carolina} MASurl sync://qewflx11.matrixone.net:30158 vaultURL
sync://qewflx10:30018/Projects/ALU selector Trunk:Latest mirrordir
/home/tbarbg8/Mirrors/Secondary/Top/ALU/ category Test type secondary
primaryserver sync://qewflx10:30148 status good}}
stcl>

In the above example, note that a pattern match was specified for the
mirror name. "ALU" by itself would not match, because it is a
submirror. And submirrors include the parent directory in their name.
```

Example Showing All Mirrors for a Specific Branch and Version

```
To find all mirrors that are mirroring the Latest version on any
branch, for the specified RS:

stcl> mirror wheremirrored sync://qewflx10:30018 -selector *:Latest
name             = Top
MASname          = North Carolina
MASurl           = sync://qewflx11.matrixone.net:30158
vaultURL         = sync://qewflx10:30018/Projects/Top
selector         = Trunk:Latest
mirrordir        = /home/tbarbg8/Mirrors/Secondary/Top/
category         = Test
type             = secondary
primaryserver    = sync://qewflx10:30148

name             = Test
MASname          = San Jose
MASurl           = sync://qewflx12.matrixone.net:30128
vaultURL         = sync://qewflx10:30018/Projects/Test
selector         = Trunk
mirrordir        = /home/tbarbg8/Mirrors/Normal/Test/
category         = Testing
type             = normal
primaryserver    =
...
```

Example Showing Finding the Primary Mirrors for a Project (File-based)

```
To find primary mirrors that are mirroring a project's vault data:

stcl> mirror wheremirrored sync://qewflx10:30018/Projects/Top \
      -type primary
name             = Top
MASname          = Cambridge
MASurl           = sync://qewflx10.matrixone.net:30148
vaultURL         = sync://qewflx10:30018/Projects/Top
selector         = Trunk:Latest
mirrordir        = /home/tbarbg8/Mirrors/Primary/Top/
category         = Development
type             = primary
primaryserver    =

succeeded 1 failed 0
stcl>
```

mirrorsetdefaultuser

mirrorsetdefaultuser

NAME

```
mirrorsetdefaultuser - Sets the default user and password using a
                       UNIX shell script
```

DESCRIPTION

```
Use this command to set the default user and password using a UNIX
shell script.

This command is subject to access controls on the server. See the
ENOVIA Synchronicity Access Control Guide for details.
```

SYNOPSIS

```
mirrorsetdefaultuser <serverURL> <defaultUser>
```

ARGUMENTS

Server URL

serverURL	Specifies the URL of the MAS SyncServer. Specify the URL as follows: sync://<host>[:<port>] where 'sync://' is required, <host> is the machine on which the SyncServer is installed, and <port> is the SyncServer port number (defaults to 2647). Example: sync://serv1.abco.com:1024

Default User

defaultUser	The default user to be used when one of the needed users is not specified for the 'mirror create' Tcl shell command. If a Tcl shell command can be run interactively, see 'mirror setoptions' for an alternate way to specify this user.

SEE ALSO

```
mirror create, mirror delete, mirror disable, mirror edit,
mirror enable, mirror get, mirror getoptions, mirror isenabled,
mirror ismirror, mirror list, mirror rename, mirror reset,
mirror setoptions, mirror status
```

EXAMPLES

```
This example sets the default user to "barbg", for the MAS
SyncServer sync://srv2.ABCo.com:2647. The user is prompted for a password.

% mirrorsetdefaultuser sync://srv2.ABCo.com:2647 barbg
Enter the password for the default user (barbg): ****

%
```

Module Cache Maintenance

Caching Objects

caching

caching Command

NAME

```
caching             - Caching behavior commands
```

DESCRIPTION

```
These commands provide a way to view and control the caching behavior
of DesignSync objects; excepting or including intellectual property
from the default caching.
```

SYNOPSIS

```
caching <caching_command>

Usage: caching disable|caching enable|caching list|caching status
```

ARGUMENTS

```
Server URL
```

RETURN VALUE

```
Various by command.
```

SEE ALSO

```
caching disable, caching enable, caching list, caching status
,
```

EXAMPLES

```
See specific command.
```

caching disable

caching disable Command

NAME

```
caching disable     - Disables object caching for server URLs
```

DESCRIPTION

```
This command disables caching for specific objects specified by
server URLs.

When object caching is disabled, the caching property of the object
URL is set to zero (0). The value of this property may be viewed with
the caching status command.

Note: Clients with this feature can disable the local caching
functionality for objects on an older (pre-3DEXPERIENCE 2016x) server
version, but older clients cannot use this feature on newer
clients. Servers accepting commands from older clients can be set up
to refuse enable/disable caching requests from older clients.  For
more information, see the DesignSync Data Manager Administrator's
Guide.

If the object for which caching is being disabled were already loaded
into a cache, those caches are not automatically removed, however
attempts to update the cache, for example with cancel, ci, populate,
or co, will fail.

This command is subject to access controls on the server.
```

SYNOPSIS

```
caching disable <SyncURL>[ <SyncURL>...]
```

ARGUMENTS

- URL

URL

```
<syncURL>  Specifies the option Sync URL as follows:
           sync://<host>[:<port>]/[<path>] or
           syncs://<host>[:<port>]/[<path>]
           where 'sync://' or 'syncs://' are required, <host> is the
           machine on which the SyncServer is installed, and <port>
           is the SyncServer port number (defaults to 2647/2679).
           And the path is the server path to the desired object.
           For example, all of these are valid syncURLs:
           sync://apollo.spaceco.com:2647
           sync://apollo.spaceco.com:2647/Modules
           sync://apollo.spaceco.com:2647/Modules/Blueprints/FuelCell2
           sync://apollo.spaceco.com:2647/Projects/SpaceShuttle
```

RETURN VALUE

```
In stcl/stclc mode, two lists are returned, where the first list
is non-empty if at least one object was successfully processed,
and the second list is non-empty if at least one object failed.
If the command failed to run, DesignSync returns an appropriate error
explaining the failure.
```

SEE ALSO

```
caching enable, caching list, caching status, url getprop, url setprop
```

EXAMPLES

- Example Showing Disabling cachability for an object

Example Showing Disabling cachability for an object

```
This example shows disabling the caching for a specific object and
verifying that the cachability was disabled using the status command,
which returns a status of zero (0).
```

```
dss> caching disable sync://srv1.ABCo.com:2647/Modules/ChipDesign/Chip
{Objects succeeded (1)} {}

dss> caching status sync://srv1.ABCo.com:2647/Modules/ChipDesign/Chip
0
```

caching enable

caching enable Command

NAME

```
caching enable      - Enables object caching for server URLs
```

DESCRIPTION

```
This command enables caching for specific objects specified by
server URLs.

When object caching is enabled, the caching property of the object
URL is set to one (1). The value of this property may be viewed with
the caching status command.

Note: Clients with this feature can enable the local caching
functionality for objects on an older (pre-3DEXPERIENCE 2016x) server
version, but older clients cannot use this feature on newer
clients. Servers accepting commands from older clients can be set up
to refuse enable/disable caching requests from older clients.  For
more information, see the DesignSync Data Manager Administrator's
Guide.

This command is subject to access controls on the server.
```

SYNOPSIS

```
caching enable <SyncURL>[ <SyncURL>...]
```

ARGUMENTS

- URL

URL

```
<syncURL>  Specifies the option Sync URL as follows:
           sync://<host>[:<port>]/[<path>] or
```

```
syncs://<host>[:<port>]/[<path>]
where 'sync://' or 'syncs://' are required, <host> is the
machine on which the SyncServer is installed, and <port>
is the SyncServer port number (defaults to 2647/2679).
And the path is the server path to the desired object.
For example, all of these are valid syncURLs:
sync://apollo.spaceco.com:2647
sync://apollo.spaceco.com:2647/Modules
sync://apollo.spaceco.com:2647/Modules/Blueprints/FuelCell2
sync://apollo.spaceco.com:2647/Projects/SpaceShuttle
```

RETURN VALUE

```
In stcl/stclc mode, two lists are returned, where the first list
is non-empty if at least one object was successfully processed,
and the second list is non-empty if at least one object failed.
If the command failed to run, DesignSync returns an appropriate error
explaining the failure.
```

SEE ALSO

```
caching disable, caching list, caching status
```

EXAMPLES

- Example Showing enabling cachability for an object

Example Showing enabling cachability for an object

```
This example shows enabling the caching for a specific object and
verifying that the cachability is enabled using the status command
which returns a status of one (1).

dss> caching enable sync://srv1.ABCo.com:2647/Modules/ChipDesign/Chip
{Objects succeeded (1)} {}

dss> caching status sync://srv1.ABCo.com:2647/Modules/ChipDesign/Chip
1
```

caching status

caching status Command

NAME

```
caching status      - Displays caching status of server URLs
```

DESCRIPTION

```
Displays the caching status (on or off) of the object URL.  URLs can
be explicitly excluded from the cache to protect access to the file
and comply with intellectual property protection needs.
```

SYNOPSIS

```
caching status <SyncURL>
```

ARGUMENTS

- URL

URL

```
<syncURL>  Specifies the option Sync URL as follows:
           sync://<host>[:<port>]/[<path>] or
           syncs://<host>[:<port>]/[<path>]
           where 'sync://' or 'syncs://' are required, <host> is the
           machine on which the SyncServer is installed, and <port>
           is the SyncServer port number (defaults to 2647/2679).
           And the path is the server path to the desired object.
           For example, all of these are valid syncURLs:
           sync://serv1.abco.com:2647
           sync://serv1.abco.com:2647/Modules
           sync://serv1.abco.com:2647/Modules/Blueprints/FuelCell2
           sync://serv1.abco.com:2647/Projects/SpaceShuttle
```

RETURN VALUE

```
Returns a value of zero (0) if the object can not be cached, or one
(1) if the object is able to be cached.
```

SEE ALSO

```
caching disable, caching enable, caching list
```

EXAMPLES

Example Showing the cachability status for an object

```
This example shows enabling/disabling the caching for a specific
object and verifying that the cachability is enabled using the status
command, which returns a zero (0) if cachability is disabled or one
(1) if cachability is enabled.

dss> caching enable sync://srv1.ABCo.com:2647/Modules/ChipDesign/Chip
{Objects succeeded (1)} {}

dss> caching status sync://srv1.ABCo.com:2647/Modules/ChipDesign/Chip
1
dss> caching disable sync://srv1.ABCo.com:2647/Modules/ChipDesign/Chip
{Objects succeeded (1)} {}

dss> caching status sync://srv1.ABCo.com:2647/Modules/ChipDesign/Chip
0
```

mcache Commands

mcache Command

NAME

```
mcache              - Module cache management commands
```

DESCRIPTION

```
The mcache commands enable an administrator to remove unused module
instances from module caches. Instances may be candidates for removal
because they are too old or not currently used. For example, if a set
of known user workspaces don't contain any mcache links to module
cache instances.

"mcache scan" searches a given list of user workspace paths looking
for module cache links to module instances within a list of provided
module cache paths. When a link to a module instance in the cache is
found, a property attached to the module cache instance is updated
with a timestamp of the current date/time. This property reflects the
last time a module cache instance was known to be referenced, and is
referred to as the "touch time".

"mcache touch" sets the "touch time" of a list of module cache
instances to the current time. "mcache touch" updates a module cache
instance timestamp property when it finds a module cache link to the
instance within one or more provided workspace paths, similar to
"cachetouchlinks" for file caches.

You may have a work flow that pre-populates a module cache. It is
```

therefore possible for a newly fetched module cache instance to have no references from a user workspace when a scrub operation is run on the mcache. As a result, new module cache entries would be prematurely removed before having the opportunity to be referenced in a user workspace. For example, suppose that an administrator has set up a weekly touch/scrub cron job. If a new module release is populated to a module cache and then scrubbed before any users have populated their workspace with the new release, it will be removed from the mcache. To prevent the new release from being scrubbed, the administrator can "mcache touch" the release.

Note that this does not consider auto-generated mirrors that populate modules into a module cache. To address this case, the populate command modifies the "touch time" of a module when its version changes. This ensures that modules freshly fetched into a module cache via a mirror have an accurate "touch time" that can be used to prevent their premature removal by a module cache scrub.

"mcache show" displays the last time module cache instances were "touched".

"mcache scrub" removes module cache instances in a list of module cache paths with a "touch time" older than a given age, similar to "cachescrubber" for file caches. Note that a module cache instance with parents cannot be removed regardless of the "touch time" unless all parents are also removed. This restriction is necessary to prevent module hierarchies from being damaged during a scrub.

The mcache commands are directly callable from a UNIX shell, similar to how the "cachescrubber" and "cachetouchlinks" commands are made available. In particular, this allows the "mcache scan" and "mcache scrub" commands to be run as cron jobs, to support automating module cache maintenance.

Note that there is no recognizable difference between a module cache and a workspace. As a result the mcache sub-commands cannot distinguish between them and therefore assume that the workspace on which they are directed to operate is in fact a module cache.

You must run "mcache scan" or "mcache touch" prior to running "mcache scrub", to ensure that active module cache instances are not inadvertently removed by "mcache scrub". This also applies to auto-generated mirrors created prior to V6R2012x, which introduced the mcache command set.

The mcache sub-commands all require one or more module cache paths on which to operate. Those that rely on a registry setting for a default list of module cache paths use SyncAdmin's General -> Modules "Default module cache paths" value.

The mcache commands are intended for administrators who have write access to the module caches. No SUID functionality is provided.

The mcache commands are only applicable to modern module caches. They cannot be used to maintain legacy module caches.

SYNOPSIS

```
mcache <mcache_command> [<mcache_command_options>]

Usage: mcache [scan|scrub|show|touch]
```

OPTIONS

```
Vary by command.
```

RETURN VALUE

```
Varies by command.
```

SEE ALSO

```
mcache scan, mcache scrub, mcache show, mcache touch
```

mcache scan

mcache scan Command

NAME

```
mcache scan         - Designates module cache instances as current
```

DESCRIPTION

- Understanding the Output

```
This command searches one or more workspace paths looking for any
module cache links within those paths to entries in any of the
supplied module cache paths. Note that the search is performed over
the root database within which the workspace path resides, looking for
module cache link instances within that root database that exist at or
below the provided path. The command will not perform a file system
search for additional root directories or symbolic links. For each
module cache link instance found the "touch time" of the entry will be
updated to the current date/time. The "touch time" is the last time a
module cache instance was known to be referenced.

Run "mcache scan" or "mcache touch" prior to running "mcache scrub",
to ensure that active module cache instances are not inadvertently
removed by "mcache scrub".
```

```
The mcache commands are only applicable to modern module caches. They
cannot be used to maintain legacy module caches.

This command supports the command defaults system.
```

Understanding the Output

```
The mcache scan command provides the option to specify the level of
information the command outputs during processing. The -report option
allows you to specify what type of information is displayed:

If you run the mcache scan command with the '-report brief' option,
the command outputs the following information:
 o Failure messages.
 o Warning messages.
 o Success/failure status.

By default, or if you run the mcache scan command with the '-report
normal' option, the command displays all the information contained in
'-report brief', and the following additional information:
 o A status message indicating that the command is scanning workspaces
   for module cache links.
 o A status message indicating that the command is gathering addresses
   of linked module cache instances.
 o A status message indicating that the command is touching module
   cache instances.

If you run the mcache scan command with the '-report verbose' option,
the command displays all the information contained in '-report normal'
and the following additional information:
 o The full URL of each module cache instance that is a candidate for
   touching.
 o The full URL of each module cache instance that was successfully
   touched.
```

SYNOPSIS

```
mcache scan [-mcachepaths <path_list>]
            [-report {brief | normal | verbose}]
            [--] <argument>[<argument>...]
```

ARGUMENTS

- Workspace Path

Workspace Path

```
<workspace path>       Path of a workspace to search for module cache
```

```
                        links. If no argument is provided the command
                        uses the current directory.
```

OPTIONS

-mcachepaths

```
 -mcachepaths <path_list>
                        Identifies one or more module caches for the
                        scan operation to update.

                        To specify multiple paths, surround the path
                        list with double quotation marks ("") and
                        separate path names with a space. For
                        example: "/dir1/cacheA /dir2/cacheB"

                        NOTES:
                        - To specify a path that includes spaces:
                          o In stcl or stclc, surround the path
                            containing the spaces with curly braces.
                            For example:
                            "/dir1/cache {/dir2/path name}".
                          o In dss or dssc, use backslashes (\) to
                            "escape" the spaces. For example:
                            "/dir1/cache /dir2/path\ with\ spaces"

                        The command scans the module caches specified
                        with the -mcachepaths option or in the default
                        module cache paths registry setting if this
                        option is not supplied. (For information about
                        the registry setting, see the "Modules Options"
                        topic in the ENOVIA Synchronicity DesignSync
                        Administrator's Guide.)
```

-report

```
 -report brief|normal|  Determines what information is returned in the
  verbose               output of the command. The information each
                        option returns is discussed in detail in the
                        "Understanding the Output" section above.

                        Valid values are:
                        o brief - outputs failures, warnings, and
                          success/failure count.

                        o normal - the default report mode, outputs
```

the command's progress. This is in addition to the information output in brief mode.

o verbose - outputs cache module instances that are being touched. This is in addition to the information output in normal mode.

--

-- Indicates that the command should stop looking for command options. Use this option when arguments to the command begin with a hyphen (-).

RETURN VALUE

In dss/dssc mode, you cannot operate on return values, so the return value is irrelevant.

In stcl/stclc mode, two lists are returned, where the first list is non-empty if at least one object was successfully processed, and the second list is non-empty if at least one object failed.

Notes:
- Scripts should only test for non-empty lists to determine success or failure. The actual content of a non-empty list currently takes the form "Objects succeeded (n)" and "Objects failed (n)", where 'n' is the number of objects that succeeded or failed. However, these return values are subject to change in a future release.
- If there are no successes and at least one failure an exception occurs (the return value is thrown, not returned).

The mcache scan UNIX command line script will return a 0 for success and a 1 for failure.

SEE ALSO

command defaults, mcache scrub, mcache show, mcache touch

EXAMPLES

In this example, "mcache scan" is run in verbose mode from UNIX, on two workspaces. The -mcachepaths option is not specified, because a default module cache path was set by the syncmgr in SyncAdmin's General -> Modules pane.

```
 % mcache scan -report verbose /home/tbarbg8/sitar /home/tbarbg7/sitar
 Logging to /home/tbarbg8/logs/dss_07202011_083658.log
 V6R2012x


 Scanning workspaces for module cache links ...
     Scanning /home/tbarbg8/sitar ...
     Scanning /home/tbarbg7/sitar ...

 Gathering addresses of linked module cache instances ...
 /home/tbarbg8/mcache/qelwsun14_30148-Alu-1.2/Alu%0: Candidate for touching
...
 /home/tbarbg8/mcache/qelwsun14_30148-Decoder-1.2/Decoder%0: Candidate for
touching ...
 /home/tbarbg8/mcache/qelwsun14_30148-Instr_reg-1.2/Instr_reg%0: Candidate
for touching ...
 /home/tbarbg8/mcache/qelwsun14_30148-Addr_calc-1.2/Addr_calc%0: Candidate
for touching ...
 /home/tbarbg8/mcache/qelwsun14_30148-Stack_pointer-1.2/Stack_pointer%0:
Candidate for touching ...
 /home/tbarbg8/mcache/qelwsun14_30148-Alu-1.2/Alu%0: Candidate for touching
...
 /home/tbarbg8/mcache/qelwsun14_30148-Decoder-1.2/Decoder%0: Candidate for
touching ...
 /home/tbarbg8/mcache/qelwsun14_30148-Instr_reg-1.2/Instr_reg%0: Candidate
for touching ...
 /home/tbarbg8/mcache/qelwsun14_30148-Addr_calc-1.2/Addr_calc%0: Candidate
for touching ...
 /home/tbarbg8/mcache/qelwsun14_30148-Stack_pointer-1.2/Stack_pointer%0:
Candidate for touching ...

 Touching module cache instances ...

 /home/tbarbg8/mcache/qelwsun14_30148-Addr_calc-1.2/Addr_calc%0: Successfully
touched.
 /home/tbarbg8/mcache/qelwsun14_30148-Alu-1.2/Alu%0: Successfully touched.
 /home/tbarbg8/mcache/qelwsun14_30148-Decoder-1.2/Decoder%0: Successfully
touched.
 /home/tbarbg8/mcache/qelwsun14_30148-Instr_reg-1.2/Instr_reg%0: Successfully
touched.
 /home/tbarbg8/mcache/qelwsun14_30148-Stack_pointer-1.2/Stack_pointer%0:
Successfully touched.
 {Objects succeeded (5)} {}
 %
```

mcache scrub

mcache scrub Command

NAME

```
 mcache scrub        - Removes old module instances from the mcache
```

DESCRIPTION

```
This command removes any module instance from the module cache that
has a "touch time" older than the supplied age. The "touch time" is
the last time a module cache instance was known to be referenced.

Note that any modules with parents will not be removed unless all of
the parents are being removed.

Run "mcache scan" or "mcache touch" prior to running "mcache scrub",
to ensure that active module cache instances are not inadvertently
removed by "mcache scrub".

The mcache commands are only applicable to modern module caches. They
cannot be used to maintain legacy module caches.

This command supports the command defaults system.
```

Understanding the Output

```
The mcache scrub command provides the option to specify the level of
information the command outputs during processing. The -report option
allows you to specify what type of information is displayed:

If you run the mcache scrub command with the '-report brief' option,
the command outputs the following information:
 o Failure messages.
 o Warning messages.
 o Success/failure status.

By default, or if you run the mcache scrub command with the '-report
normal' option, the command displays all the information contained in
'-report brief', and the following additional information:
 o A status message indicating the module cache instance being
   removed.

If you run the mcache scrub command with the '-report verbose' option,
the command displays all the information contained in '-report normal'
and the following additional information:
 o Module cache instances that are skipped, because they were touched
   within the specified number of days.
 o An informational note that all modules within the directory cone of
   each module that is a candidate for removal will also be removed.
 o Status messages for each module cache instance being removed, as
   the module content and module metadata are removed. Messages when
   the module has been deleted are also output.
```

SYNOPSIS

```
mcache scrub [-mcachepaths <path_list>]
```

```
[-report {brief | normal | verbose}] [--] <age>
```

ARGUMENTS

- Age

Age

```
<age>                  Integer value representing an age in days.
                       Module instances with a "touch time" older than
                       <age> and having no parents will be removed.
```

OPTIONS

- -mcachpaths
- -report
- --

-mcachpaths

```
-mcachepaths <path_list>
                       Identifies one or more module caches for the
                       scrub operation to process.

                       To specify multiple paths, surround the path
                       list with double quotation marks ("") and
                       separate path names with a space. For
                       example: "/dir1/cacheA /dir2/cacheB"

                       NOTES:
                       - To specify a path that includes spaces:
                         o In stcl or stclc, surround the path
                           containing the spaces with curly braces.
                           For example:
                           "/dir1/cache {/dir2/path name}".
                         o In dss or dssc, use backslashes (\) to
                           "escape" the spaces. For example:
                           "/dir1/cache /dir2/path\ with\ spaces"

                       The command processes the module caches
                       specified with the -mcachepaths option or in
                       the default module cache paths registry setting
                       if this option is not supplied. (For
                       information about the registry setting, see
                       the "Modules Options" topic in the ENOVIA
                       Synchronicity DesignSync Administrator's Guide.)
```

-report

-report brief|normal| verbose	Determines what information is returned in the output of the command. The information each option returns is discussed in detail in the "Understanding the Output" section above. Valid values are: o brief - outputs failures, warnings, and success/failure count. o normal - the default report mode, outputs a status message for each module cache instance being removed. This is in addition to the information output in brief mode. o verbose - outputs status messages are for module cache instances that are removed. Also lists cache module instances that are skipped. This is in addition to the information output in normal mode.

--

--	Indicates that the command should stop looking for command options. Use this option when arguments to the command begin with a hyphen (-).

RETURN VALUE

In dss/dssc mode, you cannot operate on return values, so the return value is irrelevant.

In stcl/stclc mode, two lists are returned, where the first list is non-empty if at least one object was successfully processed, and the second list is non-empty if at least one object failed.

Notes:
- Scripts should only test for non-empty lists to determine success or failure. The actual content of a non-empty list currently takes the form "Objects succeeded (n)" and "Objects failed (n)", where 'n' is the number of objects that succeeded or failed. However, these return values are subject to change in a future release.
- If there are no successes and at least one failure an exception occurs (the return value is thrown, not returned).

The mcache scrub UNIX command line script will return a 0 for success and a 1 for failure.

SEE ALSO

```
 command defaults, mcache scan, mcache show, mcache touch
```

EXAMPLES

```
 In this example, "mcache scrub" is run in verbose mode, to remove all
 module cache instances that haven't been touched in 120 days. The
 -mcachepaths option is not specified, because a default module cache
 path was set by the syncmgr in SyncAdmin's General -> Modules pane.

 stcl> mcache scrub -report verbose 120

 Reviewing modules in /home/tbarbg8/mcache ...
     Alu%0: Skipping; Touched within specified time.
     Decoder%0: Skipping; Touched within specified time.
     Instr_reg%0: Skipping; Touched within specified time.
     Stack_pointer%0: Skipping; Touched within specified time.

 Removing base directories for the following candidate modules:
     /home/tbarbg8/mcache/qelwsun14_30148-Addr_calc-1.2/Addr_calc%0


 Note that all modules within the directory cone of each candidate will also
be removed.

 /home/tbarbg8/mcache/qelwsun14_30148-Addr_calc-1.2/Addr_calc%0: Removing
workspace module content ...
 /home/tbarbg8/mcache/qelwsun14_30148-Addr_calc-1.2/Addr_calc%0: Removing
workspace module metadata ...
 /home/tbarbg8/mcache/qelwsun14_30148-Addr_calc-1.2/Addr_calc%0: Workspace
module removed.
 qelwsun14_30148-Addr_calc-1.2: Success deleted
 {Objects succeeded (1)} {}
 stcl>
```

mcache show

mcache show Command

NAME

```
 mcache show          - Shows the "touch time" of module cache instances
```

DESCRIPTION

- Understanding the Output

```
 This command displays the last "touch time" of module instances in one
 or more module caches. The "touch time" is the last time a module
 cache instance was known to be referenced.
```

```
The mcache commands are only applicable to modern module caches. They
cannot be used to maintain legacy module caches.

This command supports the command defaults system.
```

Understanding the Output

```
By default, or if you run the mcache show command with the "-format
text" option, the command displays a table of information for the
module cache instances found in the module cache paths.

The table includes the following information as columns:

If the mcache show command is run with the "-report brief" option:

o Instance          The module instance name.

o Available         Indicates whether the module is available for use
                    by the populate command. Possible values are "yes"
                    and "no", where a value of "yes" indicates the
                    module is available for use. A module might be
                    unavailable if, for example, it is currently being
                    fetched to the module cache.

o Last Touched      The time the module was last touched.

If the mcache show command is run with the default "-report normal"
option, then in addition to the columns listed above, these columns
are also shown:

o Name              The module name.

o Version           The version number of the module.

o Href Mode         Indicates which href mode was used to fetch the
                    module. Possible values are:
                     - dynamic - Resolves hrefs to determine what
                       version of the submodules were populated.
                     - static - Resolves hrefs to the specific
                       submodules referenced at the time the href was
                       created.
                     - normal - Resolves hrefs according to how the
                       hrefs were created. If a static href is
                       reached, the persistent mode is set to "static"
                       for that submodule and any submodules below it;
                       otherwise, the persistent mode remains
                       "normal".

                    Note: The populate command will not create an
                          mcache link to an mcached module version that
                          was not fetched statically.

o Hierarchical      Indicates whether the module was recursively
```

populated into the module mcache. Possible values are "yes" and "no", where "yes" indicates that the module was recursively populated.

If the mcache show command is run with the "-report verbose" option, then in addition to the columns shown with "-report normal", these columns are also shown:

o Selector — The selector used to fetch the module.

o Base Directory — The absolute path of the module version base directory.

o Url — The full URL of the module.

If you run the mcache show command with "-format list", it returns a Tcl list of property names and values, for each module reported.

Each module in the list has the following properties:

If the mcache show command is run with the "-report brief" option:

o modinstname — The module instance name.

o available — Indicates whether the module is available for use by the populate command. Possible values are "1" and "0", where a value of "1" indicates the module is available for use. A module might be unavailable if, for example, it is currently being fetched to the module cache.

o touched — The time the module was last touched. This is an integer value. Use "clock format" to convert the value to a recognizable date/time.

If the mcache show command is run with the default "-report normal" option, then in addition to the columns listed above, these columns are also shown:

o name — The module name.

o version — The version number of the module.

o hrefmode — Indicates which href mode was used to fetch the module. Possible values are:
- dynamic - Resolves hrefs to determine what version of the submodules were populated.
- static - Resolves hrefs to the specific submodules referenced at the time the href was created.
- normal - Resolves hrefs according to how the hrefs were created. If a static href is reached, the persistent mode is set to "static" for that submodule and any submodules below it; otherwise, the persistent mode remains "normal".

```
                    Note: The populate command will not create an
                          mcache link to an mcached module version that
                          was not fetched statically.

 o hierarchical     Indicates whether the module was recursively
                    populated into the module mcache. Possible values
                    are "1" and "0", where "1" indicates that the
                    module was recursively populated.

If the mcache show command is run with the "-report verbose" option,
then in addition to the columns shown with "-report normal", these
columns are also shown:

 o selector         The selector used to fetch the module.

 o basedir          The absolute path of the module version base
                    directory.

 o url              The full URL of the module.
```

SYNOPSIS

```
mcache show [-format {list | text}] [-mcachepaths <path_list>]
            [-report { brief | normal | verbose}]
```

OPTIONS

- -format
- -mcachepaths
- -report

-format

```
 -format list|text  Determines the format of the output.
                    Valid values are:
                    o text   Display a text table with headers and
                             columns. (Default) Objects are shown in
                             alphabetical order.

                    o list   Tcl list structure, designed for further
                             processing, and for easy conversion to a
                             Tcl array structure. This means that it
                             is a list structure in name-value pair
                             format. The top level structure is:
                             {
                                property1 <value>
                                property2 <value>
                                ...
                             }
```

```
                          A list is output for each module reported.
```

-mcachepaths

```
-mcachepaths <path_list>
                      Identifies one or more module caches for the
                      mcache show operation to process.

                      To specify multiple paths, surround the path
                      list with double quotation marks ("") and
                      separate path names with a space. For
                      example: "/dir1/cacheA /dir2/cacheB"

                      NOTES:
                      - To specify a path that includes spaces:
                        o In stcl or stclc, surround the path
                          containing the spaces with curly braces.
                          For example:
                          "/dir1/cache {/dir2/path name}".
                        o In dss or dssc, use backslashes (\) to
                          "escape" the spaces. For example:
                          "/dir1/cache /dir2/path\ with\ spaces"

                      The command processes the module caches
                      specified with the -mcachepaths option or in
                      the default module cache paths registry
                      setting if this option is not supplied. (For
                      information about the registry setting, see
                      the "Modules Options" topic in the ENOVIA
                      Synchronicity DesignSync Administrator's Guide.)
```

-report

```
-report brief|normal|  Determines what information is returned in the
 verbose               output of the command. The information each
                       option returns is discussed in detail in the
                       "Understanding the Output" section above.

                       Valid values are:
                       o brief - outputs the module instance, whether
                         it is available to link to, and the last
                         "touched time".

                       o normal - the default report mode, outputs
                         the module name, version, href mode, and
                         whether the module was fetched recursively.
                         This is in addition to the information output
                         in brief mode.

                       o verbose - outputs the module selector, base
                         directory, and full URL. This is in addition
```

```
                          to the information output in normal mode.
```

RETURN VALUE

```
 An empty string.
```

SEE ALSO

```
 command defaults, mcache scan, mcache scrub, mcache touch
```

EXAMPLES

```
 In this example, "mcache show" is run in the default "-report normal"
 mode. The -mcachepaths option is not specified, because a
 default module cache path was set by the syncmgr in SyncAdmin's
 General -> Modules pane.

 stcl> mcache show

 Name     Instance   Version  Href Mode  Available  Hierarchical  Last
Touched
 ------------------------------------------------------------------------
-----
 Alu      Alu%0      1.2      static     yes        yes           03/13/2011
03:06
 Decoder  Decoder%0  1.2      static     yes        yes           07/20/2011
13:38

 stcl>
```

mcache touch

mcache touch Command

NAME

```
 mcache touch        - Updates the "touch time" of a module instance
```

DESCRIPTION

- Understanding the Output

```
 This command updates the "touch time" of a module instance to the
 current date/time. The "touch time" is the last time a module cache
 instance was known to be referenced.
```

```
Run "mcache touch" or "mcache scan" prior to running "mcache scrub",
to ensure that active module cache instances are not inadvertently
removed by "mcache scrub".

The mcache commands are only applicable to modern module caches. They
cannot be used to maintain legacy module caches.

This command supports the command defaults system.
```

Understanding the Output

```
The mcache touch command provides the option to specify the level of
information the command outputs during processing. The -report option
allows you to specify what type of information is displayed:

If you run the mcache touch command with the '-report brief' option,
the command outputs the following information:
 o Failure messages.
 o Warning messages.
 o Success/failure status.

By default, or if you run the mcache touch command with the '-report
normal' option, the command displays all the information contained in
'-report brief', and the following additional information:
 o A progress message that module cache instances are being touched.

If you run the mcache touch command with the '-report verbose' option,
the command displays all the information contained in '-report normal'
and the following additional information:
 o A status message with the full URL of each module cache instance
   that is being touched.
```

SYNOPSIS

```
mcache touch [-report {brief | normal | verbose}] [--]
             <argument>[<argument>...]
```

ARGUMENTS

- Workspace Module

Workspace Module

```
<workspace module>     The module whose "touch time" will be updated.
                       This can be a workspace module name (if it is
                       unique in the workspace), a full workspace URL,
                       or a module instance.
```

OPTIONS

- -report
- --

-report

```
-report brief|normal|  Determines what information is returned in the
 verbose               output of the command. The information each
                       option returns is discussed in detail in the
                       "Understanding the Output" section above.

                       Valid values are:
                       o brief - outputs failures, warnings, and
                         success/failure count.

                       o normal - the default report mode, outputs
                         the command's progress. This is in addition
                         to the information output in brief mode.

                       o verbose - outputs cache module instances that
                         are being touched. This is in addition to the
                         information output in normal mode.
```

--

```
--                     Indicates that the command should stop looking
                       for command options. Use this option when
                       arguments to the command begin with a hyphen
                       (-).
```

RETURN VALUE

```
In dss/dssc mode, you cannot operate on return values, so the return
value is irrelevant.

In stcl/stclc mode, two lists are returned, where the first list
is non-empty if at least one object was successfully processed,
and the second list is non-empty if at least one object failed.

Notes:
 - Scripts should only test for non-empty lists to
   determine success or failure. The actual content of a non-empty
   list currently takes the form "Objects succeeded (n)" and
   "Objects failed (n)", where 'n' is the number of objects that
   succeeded or failed. However, these return values are subject to
   change in a future release.
 - If there are no successes and at least one failure an exception
   occurs (the return value is thrown, not returned).
```

```
The mcache touch UNIX command line script will return a 0 for success
and a 1 for failure.
```

SEE ALSO

```
command defaults, mcache scan, mcache scrub, mcache show
```

EXAMPLES

```
In this example, "mcache touch" is run in verbose mode on a module
cache instance. The full workspace URL of the module instance is
specified.

stcl> mcache touch -report verbose \
stcl> /home/tbarbg8/mcache/qelwsun14_30148-Alu-1.2/Alu%0

Touching module cache instances ...

/home/tbarbg8/mcache/qelwsun14_30148-Alu-1.2/Alu%0: Successfully touched.
{Objects succeeded (1)} {}
stcl>
```

Events and Triggers

Events

event

event Commands

NAME

```
event               - Event commands
```

DESCRIPTION

```
These commands are used to manipulate custom events, which may be
used to fire triggers.
```

SYNOPSIS

```
event <event_command> [<event_command_options>]

Usage: event [create]
```

OPTIONS

```
Vary by command.
```

RETURN VALUE

```
Varies by command.
```

SEE ALSO

```
stcl, event create, event_prop, trigger, trigger fire, trigger list
```

event create

event create Command

NAME

```
event create        - Creates a custom event
```

DESCRIPTION

```
Creates an event which may be used in conjunction with the 'trigger
list' and 'trigger fire' commands to match and execute triggers.

Events are generally created by the system, and the matching triggers
automatically executed. This command is useful primarily to exercise
triggers for testing and to extend the system by creating new kinds
of events.
```

SYNOPSIS

```
event create <name_value_list>
```

OPTIONS

- Name/Valueist

Name/Valueist

```
name_value_list
    A Tcl list of the form {name1 value1 name2 value2 ...} giving
```

the names and values of all properties that define the event. Use the 'event_prop list' command to see the list of valid property names. 'event_prop create' can be used to create new event properties.

RETURN VALUE

A new event object on success, an error if an invalid property was named.

EXAMPLES

Suppose you want to test your triggers for the 'tag' command when the tag is 'GOLDEN'.

```
set event [event create {command tag tag GOLDEN}]
foreach trigger [trigger list -event $event] {
   trigger fire $trigger $event
}
```

Here's an example of creating your own event type. Suppose you want to automate some tasks whenever a piece of your design is ready for test. Further suppose that the way you know that something is ready for test is the application of the tag 'READY_FOR_TEST'.

You could create all of your triggers like this:

```
trigger create task1 \
        -require type preObject \
        -require command tag \
        -require tag READY_FOR_TEST \
        -exec "task1 $objURL"

... repeat for each task ...
```

Unfortunately, if you change the way you indicate readiness for test, you will need to re-register all of your tasks. As an alternative, you could register a single trigger that generates 'readyForTest' events, like so:

```
trigger create FireTestTriggers \
   -require type preObject \
   -require command tag \
   -require tag READY_FOR_TEST \
   -tcl_script {
       set event [event create {type readyForTest objUrl $objURL}]
       foreach t [trigger list -event $event] {
          trigger fire $t $event
       }
   }
```

Now you can register each task for the 'readyForTest' event:

```
   trigger create task1 -require type readyForTest -exec "task1 $objURL"
   trigger create task2 -require type readyForTest -exec "task1 $objURL"
   trigger create task3 -require type readyForTest -exec "task1 $objURL"
   ... etc. ...
```

When you change how you know you're ready for test, you simply update the 'FireTestTriggers' trigger, and the individual task triggers can remain unchanged.

SEE ALSO

stcl, event_prop, trigger, trigger fire, trigger list

event_prop

event_prop Commands

NAME

event_prop - Event_prop commands

DESCRIPTION

These commands are used to create, delete, and get information about the properties that can be associated with events.

SYNOPSIS

```
event_prop <event_prop_command> [<event_prop_command_options>]

Usage: event_prop [create|delete|get|list]
```

OPTIONS

Vary by command.

RETURN VALUE

Varies by command.

SEE ALSO

```
stcl, event create, event_prop create, event_prop delete,
event_prop get, event_prop list, trigger
```

event_prop create

event_prop create Command

NAME

```
event_prop create   - Defines a new event property
```

DESCRIPTION

```
Defines a new property which may then be used to create events, which
may be used in turn to fire triggers. See 'event create'.
```

SYNOPSIS

```
event_prop create <name> [-prompt <prompt_string>] [-desc <description>]
                         [-type <value>]
```

OPTIONS

- -desc
- -name
- -prompt
- -type

-desc

```
-desc <description>
    A more verbose description of the property used to provide more
    extensive information than the prompt string. If not provided,
    the prompt string is used.
```

-name

```
name
    The name of the new event property. Must be unique among set of
    known event properties.
```

-prompt

```
-prompt <prompt_string>
    A concise description of the property used to prompt the user for
    values. If no prompt is provided, the property name is used
    instead.
```

-type

```
-type <value>
    Defines how the value of this property is passed between DesignSync
    and a trigger.
    Available -type values are:
    in    - the value of property is passed from DesignSync to the trigger
            (this is the default)
    out   - the value is passed out of trigger code to DesignSync
    inOut - the value is passed in and out.
```

SEE ALSO

```
stcl, event create, event_prop delete,
event_prop get, event_prop list, trigger
```

event_prop delete

event_prop delete Command

NAME

```
event_prop delete   - Deletes an event property definition
```

DESCRIPTION

```
Delete a previously-defined event property definition. Note that
system-defined event property definitions may not be deleted.
```

SYNOPSIS

```
event_prop delete <name>
```

OPTIONS

- name

name

```
name
    The name of the event property to be deleted.
```

SEE ALSO

```
stcl, event create, event_prop create, event_prop get,
event_prop list, trigger
```

event_prop get

event_prop get Command

NAME

```
event_prop get      - Gets information about an event property definition
```

DESCRIPTION

```
Gets the name, prompt, and description of an event property.
```

SYNOPSIS

```
event_prop get <name>
```

OPTIONS

- name

name

```
name
    The name of the event property definition being queried.
```

RETURN VALUE

```
A list of names and values suitable for array construction via the Tcl
array set command. The following attributes are defined:
```

```
name
    The name of the event property.

prompt
    The prompt string for the event property.

desc
    The verbose description of the event property.
```

SEE ALSO

```
stcl, event create, event_prop create, event_prop delete,
event_prop list, trigger
```

EXAMPLES

```
To list the attributes of all event properties:

  foreach prop [event_prop list] {
      puts "$prop:"
      array set attrArray [event_prop get $prop]
      foreach attr [array names attrArray] {
          puts "\t$attr = $attrArray($attr)"
      }
  }
```

event_prop list

event_prop list Command

NAME

```
event_prop list     - Lists known event property definitions
```

DESCRIPTION

```
Lists all event property definitions whose name match the given
expression(s). If no arguments are given, all event properties
are listed.
```

SYNOPSIS

```
event_prop list [<expr> ...]
```

OPTIONS

- expr_option

```
expr
    This argument is a regular expression used to limit the list of
    event properties returned. Multiple expressions may be
    given. If no expressions are given, all known event properties
    are returned.
```

RETURN VALUE

```
A list of the names of the matching event property definitions.
```

SEE ALSO

```
stcl, event create, event_prop create, event_prop delete,
event_prop get, trigger
```

EXAMPLES

```
To list all event properties:

    event_prop list

To list all properties with 'obj' in their names:

    event_prop list *obj*

To list all properties with 'obj' or 'URL' in their names:

    event_prop list *obj* *URL*
```

Triggers

trigger

trigger Commands

NAME

```
trigger             - Trigger commands
```

DESCRIPTION

```
These commands are used to create, delete, enable, disable, fire, and
get information about triggers.
```

SYNOPSIS

```
trigger <trigger_command> [<trigger_command_options>]

Usage: trigger [block|create|delete|disable|enable|fire|get|
                isEnabled|list|status|unblock]
```

OPTIONS

```
Vary by command.
```

RETURN VALUE

```
Varies by command.
```

SEE ALSO

```
stcl, event, event_prop, trigger create, trigger delete,
trigger disable, trigger enable, trigger fire, trigger get,
trigger isEnabled, trigger list, trigger status
```

EXAMPLES

```
See specific "trigger" commands.
```

trigger block

trigger block Command

NAME

```
trigger block       - Prevents recursive trigger script activation
```

DESCRIPTION

```
This command is used within a trigger script to indicate that the
script should not be invoked recursively when actions taken by the
script cause the invocation of triggers.
```

Triggers are automatically unblocked after the body of code has been evaluated, regardless of how that code body terminates (e.g., error, return, or normal). You therefore do not need to use the 'trigger unblock' command with 'trigger block'.

Scripts that use the 'trigger block' and 'trigger unblock' construct may cause instability. Any code written in the following format:

```
trigger block
<perform some action>
trigger unblock
```

should be rewritten as:

```
trigger block {
<perform some action>
}
```

SYNOPSIS

```
trigger block
```

OPTIONS

none

RETURN VALUE

**

SEE ALSO

trigger unblock, server-side, rstcl

EXAMPLES

Suppose that foo.tcl is a trigger script that is executed by the server whenever anyone edits a note. Within foo.tcl, a call is made to the 'note setprops' command, which changes a property of a note.

Normally this scenario would cause the script foo.tcl to be immediately executed again because a note has changed.

```
To prevent this recursive script activation, use the trigger block
command to tell the server not to invoke the script recursively.

Following is an example of how the trigger block command might be used
within a  script (it is not a complete/functional script):
  trigger block
  note setprops $SYNC_NoteURL State closed
```

trigger create

trigger create Command

NAME

```
trigger create      - Creates or replaces a trigger
```

DESCRIPTION

```
Creates a new trigger, or replaces an existing one.

A trigger is a named action, generally a script or program, that is
run by the system when a specified event occurs in the system (like a
file is checked in, tagged, etc.).

Each event has several named properties associated with it that
describe what the system is doing. See the help for 'event'
for more information.

By specifying the values of event properties that are required for a
trigger to  execute (-require), or will prevent a trigger from firing
(-exclude), a trigger can be configured to execute only when desired.
```

SYNOPSIS

```
trigger create <name> [-exec <command_and_arguments> |
               -tcl_script <script_filename> | -tcl_file <file_name> |
               -tcl_store <file_name>] [-replace]
               [-require|exclude <name> <valueExprList> ...]
```

OPTIONS

- -exclude
- -exec
- Name
- -replace
- -require
- -tcl_file
- -tcl_script

- -tcl_store

-exclude

```
-exclude <name>  Specifies that the trigger may only fire if the
<valueExprList>  value of the named event property does not match one
                 of the regular expressions in the valueExprList. If
                the named property does not exist on the event, the
                event may still fire the trigger if all other
                criteria are satisfied.
```

-exec

```
-exec <command   The given command will be executed in a subprocess
<and_arguments>  and passed the provided arguments.

                 Notes:

                 o Before the command is executed, any variables in
                   the command line will be replaced by the named
                   event property value.

                 o To include event property names as arguments: If
                   you are using the dss or dssc command shell,
                   precede the event property name with a dollar sign
                   (using Tcl variable syntax). For example: dss>
                   -exec "xterm -e vi $objURL" If you are using the
                   stcl or stclc command shell, precede the event
                   property name with a backslash and a dollar sign
                   or put the entire argument inside curly
                   braces. For example:
                   stcl> -exec "xterm -e vi \$objURL"
                   stcl> -exec {xterm -e vi $objURL}
```

Name

```
<name>
    A unique name for the trigger.
```

-replace

```
-replace         Replace the existing trigger of the same name, if
                 any. If this option is not given, and the named
                trigger already exists, an error is generated.
```

-require

-require <name> <valueExprList>
Specifies that the trigger may only fire if the value of the named event property matches one of the regular expressions in the valueExprList. If the named property does not exist on the event, the trigger will not fire.

Note: You cannot specify populate as a value for the command event property. Use co instead.

-tcl_file

-tcl_file <file_name>
The named Tcl file is loaded and executed when the trigger is fired. This enables users to edit the trigger script and have their changes take effect immediately without calling trigger create again. DesignSync looks in the current directory for the specified file; if the file is in another directory, use a full pathname for the <file_name>.

-tcl_script

-tcl_script <script>
The given script will be evaluated by the system directly.

-tcl_store

-tcl_store <file_name>
The named Tcl file is loaded immediately and the contents stored for later execution when the trigger is fired. This option is essentially the same as the -tcl_script option, except that the script is read from the file instead of the command arguments. Note that trigger create must be called again after any changes to the trigger script, otherwise the stored version of the script will not be updated.

RETURN VALUE

If the command is successful, DesignSync returns an empty string (""). If the command cannot run, DesignSync throws an error message explaining the failure.

SEE ALSO

```
stcl, event, event_prop, trigger create, trigger delete,
trigger disable, trigger enable, trigger fire, trigger get,
trigger isEnabled, trigger list, trigger status
```

EXAMPLES

Example of Registering a Tcl Script with a Trigger

```
Use -tcl_script to register a Tcl script that will keep a running
log file of all .v and .vlog files that have been checked in.

    stcl> trigger create LogCheckins \
            -require objPath "*.v *.vlog" \
            -require command "ci" \
            -require type postObject \
            -tcl_script {
                set fd [open [glob ~/checkin.log] a]
                puts $fd "$objURL"
                close $fd
            }
```

Examples of Running a Program on the Selected Files with a Trigger

```
Use -exec to run the program 'lint' on all .c files before checking
them in, except for when user 'zeus' is the one doing the checkin:
In stcl:
    stcl> trigger create LintCheck \
            -require objPath *.c \
            -require type preObject \
            -require command ci \
            -exclude user zeus \
            -exec "lint \$objPath"

Note: There are alternative formats for how you can specify command
using the -exec option.  In stcl you can also use:
         -exec {lint $objPath}

In dss mode, you can use:
         -exec "lint $objPath"
```

Example of Registering a Tcl File with a Trigger

```
Use -tcl_file to register a Tcl file to be executed after every 'tag'
```

```
command that uses the tag 'GOLDEN':

   stcl> trigger create GoldenTag \
            -require command tag \
            -require tag GOLDEN \
            -require type postCommand \
            -tcl_file goldenTag.tcl

 Note: Unlike when tcl_store is used, the tcl file is not processed
 until the trigger runs.
```

Example of Using Loading and Storing a TCL File with a Trigger

```
Use -tcl_store to immediately read and store the Tcl contained within
a file for later execution before each checkin command:

   stcl> trigger create beforeCheckin \
            -require command ci \
            -require type preCommand \
            -tcl_store preCheckin.tcl
```

trigger delete

trigger delete Command

NAME

```
trigger delete      - Deletes an existing trigger
```

DESCRIPTION

```
Deletes an existing trigger.
```

SYNOPSIS

```
trigger delete <name>
```

OPTIONS

- Name

Name

```
name     The name of the trigger to be deleted.
```

RETURN VALUE

```
1 on success, error if the trigger does not exist.
```

SEE ALSO

```
stcl, event, event_prop, trigger create, trigger delete,
trigger disable, trigger enable, trigger fire, trigger get,
trigger isEnabled, trigger list, trigger status
```

EXAMPLES

```
Create a trigger named 'foo', then delete it.

    trigger create foo -tcl_script { puts "foo" }
    trigger delete foo
```

trigger disable

trigger disable Command

NAME

```
trigger disable     - Prevents a trigger from firing
```

DESCRIPTION

```
The named trigger is marked as disabled, and will not be fired by the
system. It can be subsequently re-enabled by using the 'trigger
enable' command.

Use 'trigger status' or 'trigger isEnabled' to determine whether or
not a trigger is currently disabled.
```

SYNOPSIS

```
trigger disable <name>
```

OPTIONS

- Name

Name

```
name     The name of the trigger to be disabled.
```

RETURN VALUE

```
1 on success, error if the trigger does not exist.
```

SEE ALSO

```
stcl, event, event_prop, trigger create, trigger delete,
trigger disable, trigger enable, trigger fire, trigger get,
trigger isEnabled, trigger list, trigger status
```

EXAMPLES

```
Create a trigger named 'foo', then disable it.

    trigger create foo -tcl_script { puts "foo" }
    trigger disable foo

The stcl client returns:
```

trigger enable

trigger enable Command

NAME

```
trigger enable      - Makes a disabled trigger active again
```

DESCRIPTION

```
The named trigger is marked as enabled, and will be fired by the
system if an event matching its firing criteria is created.

Use 'trigger status' or 'trigger isEnabled' to determine whether or
not a trigger is currently enabled.
```

SYNOPSIS

```
trigger enable <name>
```

OPTIONS

- Name

Name

```
name     The name of the trigger to be enabled.
```

RETURN VALUE

```
1 on success, error if the trigger does not exist.
```

SEE ALSO

```
stcl, event, event_prop, trigger create, trigger delete,
trigger disable, trigger enable, trigger fire, trigger get,
trigger isEnabled, trigger list, trigger status
```

EXAMPLES

```
Create a trigger named 'foo', and disable it.

    trigger create foo -tcl_script { puts "foo" }
    trigger disable foo

The stcl client returns:
    1

Re-enable the trigger named foo.

    trigger enable foo

The stcl client returns:
    # 1
```

trigger fire

trigger fire Command

NAME

```
trigger fire        - Executes a trigger
```

DESCRIPTION

```
Execute the named trigger. Note that even disabled triggers may be
executed by the 'trigger fire' command; this is useful to test
triggers before enabling them.
```

SYNOPSIS

```
trigger fire <name> <event>
```

OPTIONS

Event

```
event
    An event returned by the 'event create' command.
```

Name

```
name     The name of the trigger to be executed.
```

RETURN VALUE

```
1 if the trigger succeeded, 0 if it returned an error.
```

SEE ALSO

```
stcl, event, event_prop, trigger create, trigger delete,
trigger disable, trigger enable, trigger fire, trigger get,
trigger isEnabled, trigger list, trigger status
```

EXAMPLES

```
Create a test trigger named 'echo' intended to output the names of all
non-filter events as they execute, then test it by creating a simple
event and calling trigger fire.

    trigger create echo \
```

```
            -exclude type *Filter \
            -tcl_script {
                puts "$trigger"
            }

    # create an event with the single property 'type' set to
    # 'testEvent'
    set event [event create {type testEvent}]

    # fire the trigger
    trigger fire echo $event
```

trigger get

trigger get Command

NAME

```
trigger get         - Gets information about a trigger
```

DESCRIPTION

```
Returns a Tcl list of the names and values of various attributes of
the named trigger. This command is primarily useful for use in Tcl
scripts that manage triggers. For a user-friendly display of trigger
information, use the 'trigger status' command.
```

SYNOPSIS

```
trigger get <name>
```

OPTIONS

- Name

Name

```
name     The name of the trigger to be queried.
```

RETURN VALUE

```
A Tcl list of name/value pairs of the form:
    {name1 value1 name2 value2 ...}
This list can be converted into an array using 'array set'.
The following attributes will be returned, as appropriate:
```

```
name        - name of the trigger
type        - exec, tcl_script, tcl_file, or tcl_store
fileName    - name of file for tcl_file or tcl_store triggers
commandLine - command line for exec triggers
tclScript   - the script for tcl_script or tcl_store triggers
reqProps    - required properties list
exclProps   - exclude properties list
```

SEE ALSO

```
stcl, event, event_prop, trigger create, trigger delete,
trigger disable, trigger enable, trigger fire, trigger get,
trigger isEnabled, trigger list, trigger status
```

EXAMPLES

```
Print the name and type of all known triggers:

    foreach trigger [trigger list -all] {
      array set props [trigger get $trigger]
      puts "Trigger $props(name)"
      puts "  type = $props(type)"
    }
```

trigger isEnabled

trigger isEnabled Command

NAME

```
trigger isEnabled   - Determines whether or not a trigger is enabled
```

DESCRIPTION

```
Queries the status of the named trigger, returning a 1 if the trigger
is enabled.
```

SYNOPSIS

```
trigger isEnabled <name>
```

OPTIONS

Name

```
name     The name of the trigger to be queried.
```

RETURN VALUE

```
1 if enabled, 0 if disabled, error if the trigger does not exist.
```

SEE ALSO

```
stcl, event, event_prop, trigger create, trigger delete,
trigger disable, trigger enable, trigger fire, trigger get,
trigger isEnabled, trigger list, trigger status
```

EXAMPLES

```
See if the trigger 'LintCheck' is enabled:

    if [trigger isEnabled LintCheck] {
        puts "Enabled"
    } else {
        puts "Disabled"
    }
```

trigger list

trigger list Command

NAME

```
trigger list        - Gets a list of triggers
```

DESCRIPTION

```
Lists triggers that match the given criteria. If no arguments are
given, all triggers are listed.
```

SYNOPSIS

```
trigger list [-all | -disabled | -enabled] [-event event]
```

```
[name_expr ...]
```

OPTIONS

- -all
- -disabled
- -enabled
- -event
- Name Expression

-all

```
-all            If given, lists all triggers whether they are
                enabled or disabled.

                This option is mutually exclusive with -disabled and
                -enabled.
```

-disabled

```
-disabled       If given, lists only trigger that have been
                disabled.

                This option is mutually exclusive with -all and
               -enabled.
```

-enabled

```
-enabled        If given, lists only triggers that have not been
                disabled (Default.)

                This option is mutually exclusive with -all and
                -disabled.
```

-event

```
-event event    The event given is an object returned by the 'event
                create' function. If given, only triggers which
               would be fired for the given event are listed. This
               parameter is typically used without any other flags
               to determine the set of triggers to fire for an
               event.
```

Name Expression

```
name_expr ...    Each name_expr gives a regular expression which is
                 matched against the set of all known triggers. Only
                 triggers with matching names are listed.
```

RETURN VALUE

```
A list of matching triggers.
```

SEE ALSO

```
stcl, event, event_prop, trigger create, trigger delete,
trigger disable, trigger enable, trigger fire, trigger get,
trigger isEnabled, trigger list, trigger status
```

EXAMPLES

Example of Listing All Active Triggers

```
To list all active (non-disabled) triggers:

    trigger list
```

Example of Listing All Disabled Triggers

```
To get a list of all triggers that are currently disabled:

    trigger list -disabled
```

Example of Listing All Triggers that Match a Wildcard List

```
List all enabled and disabled triggers that start with the letter
'a' or the letter 'c':

    trigger list -all a* c*
```

Example of Using Trigger List in a Script

```
Here's how this function could be used in conjunction with 'trigger
fire' to run all triggers that match an event, exiting the loop if
any error are encountered.

    # first create an event - pretend we are the ci command
    set event(type) preCommand
    set event(command) ci
    set event(objPath) foo.v
    set e [event create [array get event]]

    # use trigger list to loop all trigger that match our event
    foreach trigger [trigger list -event $e] {
        if ![trigger fire $trigger $e] {
            error "trigger $trigger failed"
        }
    }
```

trigger status

trigger status Command

NAME

```
trigger status      - Shows the status of triggers
```

DESCRIPTION

```
Prints a user-friendly list of triggers, whether or not they are
enabled, what type they are, and more information based on their type.
```

SYNOPSIS

```
trigger status [<name_expr> ...]
```

OPTIONS

- Name Expression

Name Expression

```
name_expr ...    Each name_expr gives a regular expression which is
                 matched against the set of all known triggers. Only
                 triggers with matching names are listed. If no
                 expressions are given, all known triggers are
```

```
                listed.
```

SEE ALSO

```
stcl, event, event_prop, trigger create, trigger delete,
trigger disable, trigger enable, trigger fire, trigger get,
trigger isEnabled, trigger list
```

EXAMPLES

Example of Listing Information about All Triggers

```
To list information about all triggers:

    trigger status
```

Example of Listing Information for Triggers that Match a Wildcard List

```
List information about all triggers with names that start with the
letter 'a' or the letter 'c':

    trigger status a* c*
```

Example of Listing Trigger Information for a Specific Trigger

```
List information about a trigger named 'foo':

    trigger status foo
```

trigger unblock

trigger unblock Command

NAME

```
trigger unblock     - Deprecated command
```

DESCRIPTION

```
This command was used to allow a trigger script to be recursively
executed when it created events for which it was a registered
trigger. This command is deprecated because this behavior is
the default for trigger scripts.

This command was sometimes used to undo the effect of a
previously executed 'trigger block' command. Under the new
architecture, scripts that use the 'trigger block' and 'trigger
unblock' construct may cause instability. Any code written in
the following format:

trigger block
<perform some action>
trigger unblock

should be rewritten as:

trigger block {
<perform some action>
}
```

SYNOPSIS

```
trigger unblock
```

OPTIONS

```
none
```

RETURN VALUE

```
none
```

SEE ALSO

```
trigger block, server-side, rstcl
```

Registry File Management

SyncAdmin

SyncAdmin

NAME

```
SyncAdmin           - Synchronicity Administrator tool
```

DESCRIPTION

```
Synchronicity's SyncAdmin tool is a graphical user interface that
lets system administrators, project leaders, and users configure
DesignSync clients (command-line and graphical) for
site, project, or individual use.

You execute SyncAdmin from your operating system shell, not
from a Synchronicity client shell (dss/dssc/stcl/stclc).  On
Windows platforms, you invoke SyncAdmin from the Windows
Start menu, typically:
  Start->Programs->Dassault Systems <version>->SyncAdmin

See SyncAdmin help for details on SyncAdmin. From the GUI, click
the Help button on any SyncAdmin page.
```

SYNOPSIS

```
SyncAdmin [-file <filename> | -project | -site | -user]
```

OPTIONS

-file

```
-file <filename>  Edit the specified registry file, bypassing the initial
                  SyncAdmin window (where you select which registry
                  file to edit).
```

-project

```
-project          Edit the project registry file, bypassing the initial
                  SyncAdmin window (where you select which registry
                  file to edit).
```

-site

```
-site            Edit the site registry file, bypassing the initial
                 SyncAdmin window (where you select which registry
                 file to edit).
```

-user

```
-user            Edit the user registry file, bypassing the initial
                 SyncAdmin window (where you select which registry
                 file to edit).
```

RETURN VALUE

```
none
```

SEE ALSO

```
DesSync
```

EXAMPLES

```
This example invokes SyncAdmin:
  % SyncAdmin

This example invokes SyncAdmin, in background mode, to edit the
user registry:
  % SyncAdmin -user &
```

sregistry

sregistry Commands

NAME

```
sregistry          - SyncAdmin file registry commands
```

DESCRIPTION

```
The sregistry commands allow you to view and edit the Synchronicity
Administrator registries from the command line.
```

SYNOPSIS

```
sregistry <sregistry_command> [<sregistry_command_options>]

Usage: sregistry [delete|get|keys|reset|scope|set|source|values]
```

OPTIONS

```
Vary by command.
```

RETURN VALUE

```
Varies by command.
```

SEE ALSO

```
sregistry delete, sregistry get, sregistry keys, sregistry reset,
sregistry set, sregistry scope, sregistry source, sregistry values
```

EXAMPLES

```
See specific "sregistry" commands.
```

sregistry delete

sregistry delete Command

NAME

```
sregistry delete    - Delete registry key or value
```

DESCRIPTION

```
This command deletes keys and values associated with selected
SyncAdmin registry files.  This command will not delete
read-only registry files.

After you run the "sregistry delete" command, be sure to run the
"sregistry reset" command to update the registry file. You can do this
from the client for client registry files, or in a server-side script
```

```
for the server's registry files. You can also do this by restarting
the client or server applications.
```

SYNOPSIS

```
Client-Side Invocation
sregistry delete [-currentuser | -localmachine | -synch ]
                 [-file <filename> | -project | -site | -user]
                 <keyPath> <value>

Server-Side Invocation
sregistry delete [-currentuser | -localmachine | -synch ]
                 [-file <filename> | -port | -site]
                  <keyPath> <value>
```

OPTIONS

- -currentuser
- -file
- keyPath
- -localmachine
- -port
- -project
- -site
- -synch
- -user
- value

-currentuser

```
-currentuser          Directs the command to HKEY_CURRENT_USER by adding
                      a prefix to the key 'KeyPath' with
                      "HKEY_CURRENT_USER\Software\Synchronicity"

                      This option is mutually exclusive with
                      -localmachine and -synch.
```

-file

```
-file <filename>      Executes the command using only the registry file
                      'Filename'. No other files, including
                     SyncRegistry.reg, are read. You must have write
                     permission for the specified file.

                      When invoked from the client-side, this option
                      is mutually exclusive with -project, -site, and
                      -user.  When invoked from a server-side script,
                      this option is mutually exclusive with -port,
```

and -site.

keyPath

<keyPath> Specifies the key or partial key where the registry value lives. If more than one hierarchical level is specified in the path, the syntax is very important.

In an stclc shell, the KeyPath must be enclosed in double quotes and the path elements delimited with two backslashes:
"General\\Options"
or, the KeyPath must be enclosed in braces and the path elements delimited with one backslash:
{General\Options}.

In a dssc shell, the KeyPath must be enclosed in double quotes and the path elements delimited with one backslash:
"General\Options"

If the root is not specified in the KeyPath or with the -currentuser or -localmachine options, HKEY_CURRENT_USER is searched.

-localmachine

-localmachine Directs the command to HKEY_LOCAL_MACHINE by adding a prefix to the key 'KeyPath' with "HKEY_LOCAL_MACHINE\Software\Synchronicity"

This option is mutually exclusive with -currentuser and -synch.

-port

-port This is the default context and executes the command
using the registry context: PortRegistry.reg, SiteRegistry.reg, EntRegistry.reg, SyncRegistry.reg. The -port option is only valid when called from a server side script.

This option is mutually exclusive with -file and -site.

-project

-project	Executes the command using the registry context: ProjectRegistry.reg, SiteRegistry.reg, EntRegistry.reg, SyncRegistry.reg. If the project registry is not available because $SYNC_PROJECT_CFGDIR is not defined, an error will be generated. The -project option is not allowed from a server side script as the project registry is only valid from a client tool. Requires write permission to ProjectRegistry.reg. This option is mutually exclusive with -file, -site, -user.

-site

-site	Executes the command using the registry context: SiteRegistry.reg, EntRegistry.reg, SyncRegistry.reg. Requires write permission to SiteRegistry.reg. When invoked from the client-side, this option is mutually exclusive with -file, -project, and -user. When invoked from a server-side script, this option is mutually exclusive with -file, and -port.

-synch

-synch	Directs the command to Software\Synchronicity by adding a prefix to the key 'KeyPath' with "Software\Synchronicity" This option is mutually exclusive with -currentuser and -localmachine.

-user

-user	This is the default context and executes the command using the registry context: UserRegistry.reg, ProjectRegistry.reg, SiteRegistry.reg, EntRegistry.reg, SyncRegistry.reg. The -user option is valid only when called from a client tool. Requires write permission to UserRegistry.reg. This option is mutually exclusive with -file, -project, and -site.

value

```
<value>              The name of the registry value to retrieve the
                     data from.
```

RETURN VALUE

```
Returns an empty string on success. Deleting a value that does not
exist will return an empty string. Deleting a key where the leaf
name does not exist will return an empty string.

If a Value is not specified, the key and all its values are removed.
If the key contains sub-keys, neither the key nor it values are
removed. You can not remove a key that contains subkeys.

The delete command can only delete keys and values that are in the
registry file that is open for write (the first file listed in the
registry context). If you try to delete a value that exists in
one of the read-only registry files, an error will be generated. If
you try to delete a key and the key and all its values exist in
read-only registry files, an error will be generated. If some of the
key's values are in the writable registry file, only those values
will be deleted and the command will return OK. The key and the values
in the read-only registry files will still remain.
```

SEE ALSO

```
sregistry get, sregistry set, sregistry keys, sregistry reset,
sregistry source, sregistry values
```

EXAMPLES

```
Continuing from the 'sregistry set' example, which showed a user setting
their own default fetch state, overriding the site-wide "share" preference.

stcl> sregistry set -currentuser -user "General\\Options" \
stcl> DefaultFetchType get
get
stcl>

The user can remove their default fetch state setting:

stcl> sregistry delete -currentuser -user "General\\Options"
DefaultFetchType
stcl>
```

sregistry get

sregistry get Command

NAME

```
sregistry get       - Get a registry value
```

DESCRIPTION

```
Retrives the value of a registry key from the specified registry. The
complete list of available registry keys is contained in the ENOVIA
Synchronicity DesignSync Administrator's Guide.
```

SYNOPSIS

```
Client-side Invocation
sregistry get [-base dec | hex]
              [-currentuser | -localmachine | -synch]
              [-default <dataPath>] [-format text | list]
              [-user | -project | -site | -file <filename>]
              <keyPath> <value>

Server-side Invocation
sregistry get [-base dec | hex]
              [-currentuser | -localmachine | -synch]
              [-default <dataPath>] [-format text | list]
              [-port | -site | -file <filename>] <keyPath> <value>
```

OPTIONS

- -base
- -currentuser
- -default
- -file
- -format
- -localmachine
- -port
- -project
- -site
- -synch
- -user
- Key Path
- Value

-base

-base	Specifies how to represent numerical data. The default is dec (decimal). The -base option has no effect if the data is of type string. If the base is selected as dec, the data will be represented as a signed integer. If the base is selected as hex (hexadecimal), the data will be output in unsigned hexadecimal format starting with 0x and showing all four bytes. For example, the same registry value might output 0xffffffff in hex base and -1 in dec base.

-currentuser

-currentuser	Directs the command to HKEY_CURRENT_USER by adding a prefix to the key 'KeyPath' with "HKEY_CURRENT_USER\Software\Synchronicity" This option is mutually exclusive with -localmachine and -synch.

-default

-default <dataPath>	If a default value, <dataPath>, is specified, the command will return DefaultData if the value was not found in the registry. If a default is not specified and the value is not found in the registry, an error is returned.

-file

-file <filename>	Executes the command using only the registry file <filename>. No other files, including SyncRegistry.reg, are read. You must have write permission for the specified file. When invoked from the client-side, this option is mutually exclusive with -project, -site, and -user. When invoked from a server-side script, this option is mutually exclusive with -port, -site, and -user.

-format

-format	Specifies the way the output will be returned. The default is text. The format text will return the data from the registry as a string. If the registry

```
value is a dword, the return value is a string
representing the dword. The format list will return
a tcl list of name value pairs. The following named
values will display:

data: The data requested from the registry with the
      get command.

type:   The type of the data that was returned
        from the registry. The type will be
        either number or string.
        If the value was not found in the registry,
        and the default data was returned, the type
        will be set to string.
source: A string indicating which registry file the
        value was found in. The string will be
        one of the following: Current, Default,
        Override, or None. See the sregistry
        source command for more information.
root:   A string containing the name of the root
        (hive) where the value was found. The
        string will be either "HKEY_CURRENT_USER"
        or "HKEY_LOCAL_MACHINE".
        If the value was not found in the
        registry, and the default data was
        returned, this value will be set to
        "HKEY_CURRENT_USER".
```

-localmachine

```
 -localmachine      Directs the command to HKEY_LOCAL_MACHINE by adding
                    a prefix to the key 'KeyPath' with
                    "HKEY_LOCAL_MACHINE\Software\Synchronicity"

                    This option is mutually exclusive with
                    -currentuser and -sync.
```

-port

```
 -port              This is the default context and executes the command
                    using the registry context: PortRegistry.reg,
                    SiteRegistry.reg, EntRegistry.reg, SyncRegistry.reg.
                    The -port option is only valid when called from a
                    server side script.

                    This option is mutually exclusive with -file and
                    -site.
```

-project

-project	Executes the command using the registry context: ProjectRegistry.reg, SiteRegistry.reg, EntRegistry.reg, SyncRegistry.reg. If the project registry is not available because $SYNC_PROJECT_CFGDIR is not defined, an error will be generated. The -project option is not allowed from a server side script as the project registry is only valid from a client tool. Requires write permission to ProjectRegistry.reg. This option is mutually exclusive with -file, -site, and -user.

-site

-site	Executes the command using the registry context: SiteRegistry.reg, EntRegistry.reg, SyncRegistry.reg. Requires write permission to SiteRegistry.reg. When invoked from the client-side, this option is mutually exclusive with -file, -project, and -user. When invoked from a server-side script, this option is mutually exclusive with -file, -port, and -user.

-synch

-synch	Directs the command to Software\Synchronicity by adding a prefix to the key 'KeyPath' with "Software\Synchronicity" This option is mutually exclusive with -currentuser and -localmachine.

-user

-user	This is the default context and executes the command using the registry context: UserRegistry.reg, ProjectRegistry.reg, SiteRegistry.reg, EntRegistry.reg, SyncRegistry.reg. The -user option is only valid when called from a client tool. Requires write permission to UserRegistry.reg. This option is mutually exclusive with -file, -project, and -site.

Key Path

<keyPath>	Specifies the key or partial key where the registry value lives. If more than one hierarchical level is specified in the path, the syntax is very important. In an stclc shell, the key path must be enclosed in double quotes and the path elements delimited with two backslashes: "General\\Options" or, the key path must be enclosed in braces and the path elements delimited with one backslash: {General\Options}. In a dssc shell, the key path must be enclosed in double quotes and the path elements delimited with one backslash: "General\Options" If the root is not specified in the key path or with the -currentuser or -localmachine options, HKEY_CURRENT_USER is searched first for the value, and if it is not found, HKEY_LOCAL_MACHINE is searched.

Value

<value>	The name of the registry value to retrieve the data from.

SEE ALSO

sregistry set, sregistry keys, sregistry values, sregistry delete, sregistry source, sregistry reset

EXAMPLES

Continuing from the 'sregistry values' example, which showed:

```
stcl> sregistry values -currentuser -site "General\\Options"
DefaultFetchType
stcl>
```

To find the site-wide default fetch state value:

```
stcl> sregistry get -currentuser -site "General\\Options" DefaultFetchType
share
stcl>
```

See the 'sregistry set' example for how a user would set their own default

```
fetch state, overriding the site default value.
```

sregistry keys

sregistry keys Command

NAME

```
sregistry keys      - Displays sub-keys in registry value
```

DESCRIPTION

```
This command displays a list of the sub-keys associated with a specified
KeyPath.
```

SYNOPSIS

```
Client-side invocation:
  sregistry keys [-currentuser | -localmachine | -synch ]
                 [-user | -project | -site | -file <filename>]
                 [-format text | list] <KeyPath>

Server-side invocation:
  sregistry keys [-currentuser | -localmachine | -synch ]
                 [-format text | list]
                 [-port | -site | -file <filename>] <KeyPath>
```

OPTIONS

- -currentuser
- -file
- -format
- Key Path
- -localmachine
- -port
- -project
- -site
- -synch
- -user

-currentuser

```
-currentuser        Adds the following prefix to the key <KeyPath>:
                    "HKEY_CURRENT_USER\Software\Synchronicity"
```

This option is mutually exclusive with -localmachine, and -synch.

-file

-file <filename> Executes the command using only the registry file <filename>. No other files are read, including SyncRegistry.reg. You must have write permission for the specified file.

When invoked from the client-side, this option is mutually exclusive with -project, -site, and -user. When invoked from a server-side script, this option is mutually exclusive with -port, and -site.

-format

-format Specifies the way the output will be returned. The default is text. The format text will return each key on a new line. The format list will list the keys in a tcl list.

Key Path

<KeyPath> Specifies the top-level key path or partial key path. If more than one hierarchical level is specified in the path, the syntax is very important.

In an stclc shell, the KeyPath must be enclosed in double quotes and the path elements delimited with two backslashes:
"General\\Options"
or, the KeyPath must be enclosed in braces and the path elements delimited with one backslash:
{General\Options}.

In a dssc shell, the KeyPath must be enclosed in double quotes and the path elements delimited with one backslash:
"General\Options"

If the root is not specified in the KeyPath or with the -currentuser or -localmachine options, HKEY_CURRENT_USER is searched.

-localmachine

-localmachine	Adds the following prefix to the key <KeyPath>: "HKEY_LOCAL_MACHINE\Software\Synchronicity" This option is mutually exclusive with -currentuser and -synch.

-port

-port	Executes the command using this registry hierarchy: PortRegistry.reg, SiteRegistry.reg, EntRegistry.reg, SyncRegistry.reg. The -port option is the default. This option is only valid when called from a server-side script. This operation is mutually exclusive with -file, and -site.

-project

-project	Executes the command using this registry hierarchy: ProjectRegistry.reg, SiteRegistry.reg, EntRegistry.reg, SyncRegistry.reg. If the project registry is not available because $SYNC_PROJECT_CFGDIR is not defined, an error will be generated. The -project option is not allowed from a server side script as the project registry is only valid from a client tool. This operation is mutually exclusive with -file, -site, and -user.

-site

-site	Executes the command using this registry hierarchy: SiteRegistry.reg, EntRegistry.reg, SyncRegistry.reg. When invoked from the client-side, this option is mutually exclusive with -file, -project, and -user. When invoked from a server-side script, this option is mutually exclusive with -file, and -port.

-synch

-synch	Adds the following prefix to the key <KeyPath>: "Software\Synchronicity"

```
                    This option is mutually exclusive with
                    -currentuser and -localmachine.
```

-user

```
 -user              Executes the command using this registry hierarchy:
                    UserRegistry.reg, ProjectRegistry.reg,
                   SiteRegistry.reg, EntRegistry.reg,
                   SyncRegistry.reg. The -user option is the
                   default. This option is valid only when called
                   from a client tool.

                    This operation is mutually exclusive with -file,
                    -project, and -site.
```

RETURN VALUE

```
A list of the sub-keys of KeyPath. If no subkeys exist, then an empty
list is returned.
```

SEE ALSO

```
sregistry get, sregistry set, sregistry values, sregistry delete,
sregistry source, sregistry reset
```

EXAMPLES

```
The example below finds the site-wide General registry keys.  When an
empty string is returned, that means there are no sub-keys below the
specified KeyPath.

stcl> sregistry keys -currentuser -site General
ExtensionTypes
CmdTable
Options
stcl> sregistry keys -currentuser -site "General\\ExtensionTypes"
stcl> sregistry keys -currentuser -site "General\\Options"
stcl> sregistry keys -currentuser -site "General\\CmdTable"
DefaultLogDir
stcl>

The 'sregistry values' example shows how to use the above result.
```

sregistry reset

sregistry reset Command

NAME

```
sregistry reset     - Forces a refresh of all registry files
```

DESCRIPTION

```
This command forces the re-reading of all registry files,
including the read-only files. It returns an empty string upon
successful completion. It is important to note that reloading all the
registry files may not be sufficient to cause Synchronicity tools
(client and server) to immediately see the new settings. This is
because some values are cached by the programs. To assure all new
values are being read by an application, the application should be
restarted.
```

SYNOPSIS

```
sregistry reset
```

OPTIONS

```
None.
```

SEE ALSO

```
sregistry get, sregistry set, sregistry keys, sregistry values
sregistry source, sregistry delete
```

sregistry scope

sregistry scope Command

NAME

```
sregistry scope     - Temporarily changes which registry files are active
```

DESCRIPTION

Used only with the command defaults system, to temporarily change which registry files are active. By default, "defaults set" stores default values in the user's registry file. Use "sregistry scope" to store default values in other registry files that are sourced by the DesignSync client on startup, such as the installation's site registry file.

Within a DesignSync client session, run "defaults refresh" to read all default values from the client registry files. See the DesignSync Data Manager User's Guide topic "Registry Files" for further information.

To prevent users' saved default values from overriding site or project default values, specify the "-nooverrule" option to the "defaults set" command. See the "defaults set" command documentation for details.

SYNOPSIS

```
sregistry scope [-project | -site]
                {defaults set -- <command> <option> [<option> ...]}
```

OPTIONS

- -project
- -site

-project

```
-project        If <SYNC_PROJECT_CFGDIR> is defined, store default values
                in the <SYNC_PROJECT_CFGDIR>/ProjectRegistry.reg file.
                Requires write permission to the ProjectRegistry.reg file.
```

-site

```
-site           Store default values in the site-wide registry file,
                <SYNC_SITE_CNFG_DIR>/SiteRegistry.reg. If not defined,
                <SYNC_SITE_CNFG_DIR> resolves to <SYNC_SITE_CUSTOM>/config
                which, in turn, resolves to <SYNC_CUSTOM_DIR>/site/config.
                Requires write permission to the SiteRegistry.reg file.
```

RETURN VALUE

The result of the expression given to the "sregistry scope" command.

SEE ALSO

```
defaults refresh, defaults set, defaults show, command defaults
```

EXAMPLES

```
As the installation owner, to set a default report mode for the "ls"
command, for all users at your site:

stcl> sregistry scope -site {defaults set -- ls -report verbose}

The "defaults show" command confirms that the "-report verbose" default
value for the "ls" command is set in the site registry file.

stcl> defaults show -source ls
{ls temporary {} project {} project_nooverrule {} user {} user_nooverrule
{} site {-report verbose} site_nooverrule {} enterprise {}
enterprise_nooverrule {}}
stcl>
```

sregistry set

sregistry set Command

NAME

```
sregistry set       - Sets a registry value
```

DESCRIPTION

```
The 'sregistry set' and related 'sregistry' commands allow you to edit
the Synchronicity Administrator registries from the command
line. When you enter the command, you choose the client registry
or the server registry. You also need to specify the key or
where the registry value lives (HKEY_CURRENT_USER or
HKEY_LOCAL_MACHINE). The last part of the command identifies the
KeyPath (SyncAdmin settings tree) and the key name you want to retrieve.

After you run the "sregistry delete" command, be sure to run the
"sregistry reset" command to update the registry file. You can do this
from the client for client registry files, or in a server-side script
for the server's registry files. You can also do this by restarting
the client or server applications.
```

SYNOPSIS

```
Client-Side Invocation
sregistry set [-currentuser | -localmachine | -synch ]
              [-file <filename> | -project | -site | -user]
```

```
              <keyPath> [-type number|string] <value> [--] Data

Server-Side Invocation
sregistry set [-currentuser | -localmachine | -synch ]
              [-file <filename> | -site | -port]
              <keyPath> [-type number|string] <value> [--] Data
```

ARGUMENTS

- Data

Data

Data	Specifies the data to write into the registry. Data must match the type specified with -type. A number can be an integer (signed or unsigned) or a hexadecimal value. Hexadecimal numbers must be prefixed with '0x' or '0X' For Example: 0xFF2A.

OPTIONS

- -currentuser
- -file
- keypath
- -localmachine
- -port
- -project
- -site
- -synch
- -type
- -user
- Value
- --

-currentuser

-currentuser	Directs the command to HKEY_CURRENT_USER by adding a prefix to the key 'KeyPath' with "HKEY_CURRENT_USER\Software\Synchronicity" This option is mutually exclusive with -localmachine and -synch.

-file

-file <filename>	Executes the command using only the registry file 'Filename'. No other files, including SyncRegistry.reg, are read. You must have write permission for the specified file. When invoked from the client-side, this option is mutually exclusive with -project, -site, and -user. When invoked from a server-side script, this option is mutually exclusive with -port, and -site.

keypath

<keyPath>	Specifies the key or partial key where the registry value lives. If more than one hierarchical level is specified in the path, the syntax is very important. In an stclc shell, the key path must be enclosed in double quotes and the path elements delimited with two backslashes: "General\\Options" or, the key path must be enclosed in braces and the path elements delimited with one backslash: {General\Options}. In a dssc shell, the key path must be enclosed in double quotes and the path elements delimited with one backslash: "General\Options" If the root is not specified in the key path or with the -currentuser or -localmachine options, HKEY_CURRENT_USER is used.

-localmachine

-localmachine	Directs the command to HKEY_LOCAL_MACHINE by adding a prefix to the key 'KeyPath' with "HKEY_LOCAL_MACHINE\Software\Synchronicity" This option is mutually exclusive with -currentuser and -synch.

-port

-port	This is the default context and executes the command using the registry context: PortRegistry.reg, SiteRegistry.reg, EntRegistry.reg, SyncRegistry.reg. The -port option is only valid when called from a server-side script.

This option is mutually exclusive with -file and -site.

-project

-project — Executes the command using the registry context: ProjectRegistry.reg, SiteRegistry.reg, EntRegistry.reg, SyncRegistry.reg. If the project registry is not available because $SYNC_PROJECT_CFGDIR is not defined, an error will be generated. The -project option is not allowed from a server side script as the project registry is only valid from a client tool. Requires write permission to ProjectRegistry.reg.

This option is mutually exclusive with -file, -site, and -user.

-site

-site — Executes the command using the registry context: SiteRegistry.reg, EntRegistry.reg, SyncRegistry.reg. Requires write permission to configuration directory for the site ($SYNC_SITE_CNFG_DIR) and the SiteRegistry.reg file.

When invoked from the client-side, this option is mutually exclusive with -file, -project, and -user. When invoked from a server-side script, this option is mutually exclusive with -file, and -port.

-synch

-synch — Directs the command to Software\Synchronicity by adding a prefix to the key 'KeyPath' with "Software\Synchronicity"

This option is mutually exclusive with -currentuser and -localmachine.

-type

-type — Specifies the type of data to store in the registry. Can be either string or number with string being the default.

-user

-user	This is the default context and executes the command using the registry context: UserRegistry.reg, ProjectRegistry.reg, SiteRegistry.reg, EntRegistry.reg, SyncRegistry.reg. The -user option is only valid when called from a client tool. Requires write permission to UserRegistry.reg. This option is musually exclusive with -file, -project, and -site.

Value

<value>	The name of the registry value to retrieve the data from.

--

--	Specifies that there are no more switches to follow on the command line.

RETURN VALUE

The value of Data is returned.

SEE ALSO

sregistry get, sregistry keys, sregistry values, sregistry delete, sregistry source, sregistry reset

EXAMPLES

Continuing from the 'sregistry get' example, which showed the site-wide default fetch state:

```
stcl> sregistry get -currentuser -site "General\\Options" DefaultFetchType
share
stcl>
```

A user can set their own default fetch state, overriding the site-wide preference.

```
stcl> sregistry set -currentuser -user "General\\Options" \
stcl> DefaultFetchType get
get
```

```
stcl>

See the 'sregistry source' example for how to determine which registry
file's default fetch state value is being used.
```

sregistry source

sregistry source Command

NAME

```
sregistry source    - Displays the source of a registry value
```

DESCRIPTION

```
This command displays the source (registry file) of a SyncAdmin
registry value.
```

SYNOPSIS

```
Client-Side Invocation
sregistry source [-currentuser | -localmachine | -synch]
                 [-file <filename> | -project | -site | -user]
                  <keyPath> <value>

Server-Side Invocation
sregistry source [-currentuser | -localmachine | -synch]
                 [-file <filename> | -port | -site]
                 <keyPath> <value>
```

OPTIONS

- -currentuser
- -file
- KeyPath
- -localmachine
- -port
- -project
- -site
- -synch
- -user
- value

-currentuser

-currentuser	Directs the command to HKEY_CURRENT_USER by adding a prefix to the key 'KeyPath' with "HKEY_CURRENT_USER\Software\Synchronicity" This option is mutually exclusive with -localmachine and -synch.

-file

-file <filename>	Executes the command using only the registry file 'Filename'. No other files, including SyncRegistry.reg, are read. You must have write permission for the specified file. When invoked from the client-side, this option is mutually exclusive with -project, -site, and -user. When invoked from a server-side script, this option is mutually exclusive with -port, and -site.

KeyPath

<keyPath>	Specifies the key or partial key where the registry value lives. If more than one hierarchical level is specified in the path, the syntax is very important. In an stclc shell, the key path must be enclosed in double quotes and the path elements delimited with two backslashes: "General\\Options" or, the key path must be enclosed in braces and the path elements delimited with one backslash: {General\Options}. In a dssc shell, the key path must be enclosed in double quotes and the path elements delimited with one backslash: "General\Options" If the root is not specified in the key path or with the -currentuser or -localmachine options, HKEY_CURRENT_USER is searched.

-localmachine

-localmachine	Directs the command to HKEY_LOCAL_MACHINE by adding a prefix to the key 'KeyPath' with "HKEY_LOCAL_MACHINE\Software\Synchronicity"

This option is mutually exclusive with -currentuser and -synch.

-port

-port — This is the default context and executes the command using the registry context: PortRegistry.reg, SiteRegistry.reg, EntRegistry.reg, SyncRegistry.reg. The -port option is only valid when called from a server-side script.

This option is mutually exclusive with -file and -site.

-project

-project — Executes the command using the registry context: ProjectRegistry.reg, SiteRegistry.reg, EntRegistry.reg, SyncRegistry.reg. If the project registry is not available because $SYNC_PROJECT_CFGDIR is not defined, an error will be generated. The -project option is not allowed from a server side script as the project registry is only valid from a client tool. Requires write permission to ProjectRegistry.reg.

This option is mutually exclusive with -file, -site, and -user.

-site

-site — Executes the command using the registry context: SiteRegistry.reg, EntRegistry.reg, SyncRegistry.reg. Requires write permission to SiteRegistry.reg.

When invoked from the client-side, this option is mutually exclusive with -file, -project, and -user. When invoked from a server-side script, this option is mutually exclusive with -file, and -port.

-synch

-synch — Directs the command to Software\Synchronicity by adding a prefix to the key 'KeyPath' with "Software\Synchronicity"

This option is mutually exclusive with -currentuser and -synch.

-user

-user	This is the default context and executes the command using the registry context: UserRegistry.reg, ProjectRegistry.reg, SiteRegistry.reg, EntRegistry.reg, SyncRegistry.reg. The -user option is only valid when called from a client tool. Requires write permission to UserRegistry.reg. This option is mutually exclusive with -file, -project, and -site.

value

<value>	The name of the registry value to retrieve the data from.

RETURN VALUE

```
Current|Default|Override|None
Returns 'Current' if the value comes from the registry file currently
open for write.
Returns 'Default' if the value comes from a registry with lower
precedence than the registry file currently open for write.
Returns 'Override' if the value comes from a registry with higher
precedence than the registry file currently open for write.
Returns 'None' if the value is not found in the registry.
```

SEE ALSO

```
sregistry get, sregistry set, sregistry keys, sregistry reset
sregistry delete, sregistry values
```

EXAMPLES

```
Continuing from the 'sregistry set' example, which showed a user setting
their own default fetch state, overriding the site-wide "share" preference.

stcl> sregistry set -currentuser -user "General\\Options" \
stcl> DefaultFetchType get
get
stcl>

To determine which registry file (the site-wide registry file or the user's
```

```
 registry file) is being sourced for the default fetch state value:

 stcl> sregistry source -currentuser -user "General\\Options"
DefaultFetchType
 Current
 stcl>

 This shows that the default fetch state value in the user's registry file
 (the registry file currently open for writing by the DesignSync client) is
 in use.
```

sregistry values

sregistry values Command

NAME

```
 sregistry values    - Displays the available registry values
```

DESCRIPTION

```
 This command displays a list of Values available under the <keyPath> key.
```

SYNOPSIS

```
 Client-Side Invocation
 sregistry values [-currentuser | -localmachine | -synch ]
                  [-user | -project | -site | -file <filename>]
                  [-port | -site | -file <filename>]
                  [-format text | list] <keyPath>

 Server-Side Invocation
 sregistry values [-currentuser | -localmachine | -synch ]
                  [-user | -project | -site | -file <filename>]
                  [-port | -site | -file <filename>]
                  [-format text | list] <keyPath>
```

OPTIONS

- -currentuser
- -file
- -format
- KeyPath
- -localmachine
- -port
- -project
- -site
- -synch

- -user

-currentuser

-currentuser	Directs the command to HKEY_CURRENT_USER by adding a prefix to the key 'KeyPath' with "HKEY_CURRENT_USER\Software\Synchronicity" This option is mutually exclusive with -localmachine and -synch.

-file

-file <filename>	Executes the command using only the registry file 'Filename'. Only the file ('Filename') will be read or written. No other files, including the SyncRegistry.reg file are read. When invoked from the client-side, this option is mutually exclusive with -project, -site, and -user. When invoked from a server-side script, this option is mutually exclusive with -port, and -site.

-format

-format	Specifies the way the output will be returned. The default is text. The format text will return each value on a new line. The format list will list the values in a tcl list.

KeyPath

<keyPath>	Specifies the key or partial key where the registry value lives. If more than one hierarchical level is specified in the path, the syntax is very important. In an stclc shell, the key path must be enclosed in double quotes and the path elements delimited with two backslashes: "General\\Options" or, the key path must be enclosed in braces and the path elements delimited with one backslash: General\Options}. In a dssc shell, the key path must be enclosed in double quotes and the path elements delimited with one backslash: "General\Options"

```
                    If the root is not specified in the key path or
                    with the -currentuser or -localmachine options,
                    HKEY_CURRENT_USER is searched.
```

-localmachine

```
 -localmachine      Directs the command to HKEY_LOCAL_MACHINE by adding
                    a prefix to the key 'KeyPath' with
                    "HKEY_LOCAL_MACHINE\Software\Synchronicity"

                    This option is mutually exclusive with
                    -currentuser and -synch.
```

-port

```
 -port              This is the default context and executes the command
                    using the registry context: PortRegistry.reg,
                    SiteRegistry.reg, EntRegistry.reg, SyncRegistry.reg.
                    The -port option is only valid when called from a
                    server-side script.

                    This option is mutually exclusive with -file and
                   -site.
```

-project

```
 -project           Executes the command using the registry context:
                    ProjectRegistry.reg, SiteRegistry.reg, EntRegistry.reg,
                    SyncRegistry.reg. If the project registry is not
                    available because $SYNC_PROJECT_CFGDIR is not defined,
                    an error will be generated. The -project option is not
                    allowed from a server side script as the project
                    registry is only valid from a client tool. Requires
                    write permission to ProjectRegistry.reg.

                    This option is mutually exclusive with -file,
                    -site, and -user.
```

-site

```
 -site              Executes the command using the registry context:
                    SiteRegistry.reg, EntRegistry.reg, SyncRegistry.reg.
                    Requires write permission to SiteRegistry.reg.

                    When invoked from the client-side, this option is
                    mutually exclusive with -file, -project, and -user.
                    When invoked from a server-side script,
```

this option is mutually exclusive with -file and -port.

-synch

-synch	Directs the command to Software\Synchronicity by adding a prefix to the key 'KeyPath' with "Software\Synchronicity" This option is mutually exclusive with -currentuser and -localmachine.

-user

-user	This is the default context and executes the command using the registry context: UserRegistry.reg, ProjectRegistry.reg, SiteRegistry.reg, EntRegistry.reg, SyncRegistry.reg. The -user option is only valid when called from a client tool. Requires write permission to UserRegistry.reg. This option is mutually exclusive with -file, -project, and -site.

RETURN VALUE

A list of the Values available under the <keyPath> key.
If no Values exist, then an empty list is returned.

SEE ALSO

sregistry get, sregistry set, sregistry keys, sregistry delete, sregistry source, sregistry reset

EXAMPLES

Continuing from the 'sregistry keys' example, which showed:

```
stcl> sregistry keys -currentuser -site General\\Options
stcl>
```

To find the registry values available below General\\Options:

```
stcl> sregistry values -currentuser -site "General\\Options"
DefaultFetchType
```

```
stcl>

This means that a default fetch state has been set site-wide.

The 'sregistry get' example shows how to retrieve that default value.
```

Server Backup

backup

backup Command

NAME

```
backup              - Backs up a SyncServer
```

DESCRIPTION

```
The backup command lets you back up a SyncServer, including its vault,
metadata, and notes. You can use this command to run both full and
incremental backups. Use the backup command in conjunction with
your standard system back-up procedures. To create a back-up that
lets you safely restore your server, you take two steps:

1. Use the backup command to generate back-up data in the
   server_vault/Backup.sync area.
2. Perform a standard system back-up of your server_vault area and
   your $SYNC_CUSTOM_DIR.

The data stored in the Backup.sync directory creates a server
snapshot that, in conjunction with a standard system back-up, lets
you safely restore your server and vault.

The backup command is server-side only and must be run using the rstcl
command or by passing a script in a URL from your browser. See
the 'server-side' topic or the ENOVIA Synchronicity stcl Programmer's
Guide for details.

The back-up operation includes the server vault, metadata,
attachments, and all of the notes, note types, and property types.
The back-up preserves the following directories:

   o The entire server_metadata hierarchy
   o The entire server_vault hierarchy
   o Parts of the $SYNC_CUSTOM_DIR/servers/<host>/<port>/share/data
     hierarchy.

Only attachments are copied from the $SYNC_CUSTOM_DIR areas.
Note customizations stored in the custom areas are not
backed up.
```

The backup command does not compress the backed-up data.

The data is backed up to the directory:

  $SYNC_DIR/../syncdata/<host>/<port>/server_vault/Backup.sync

This directory contains a subdirectory for each back-up operation. The back-up subdirectory name has the following format:

  <year><month><day>_<hour><minute><second>

The time is in 24-hour format according to the local time zone of the server.

Within the dated subdirectory, the backed-up data is stored in the following directories:

- Attachments - Hard links to the note attachment and definition files in the server's $SYNC_CUSTOM_DIR/servers/<host>/<port>/share/data area.
- server_metadata/ and all directories below it - Copies of the metadata.
- server_vault/ and all directories below it - Copies of the tags database are stored in the server_vault/Partitions subdirectories. Hard links to the server vault are stored in the server_vault directory.

For example, data in:

  $SYNC_DIR/../syncdata/<host>/<port>/server_vault/Projects/P1/file1.rca

might be backed up to:

  $SYNC_DIR/../syncdata/<host>/<port>/server_vault/Backup.sync/20010710_145601/Projects/P1/file1.rca

The Backup.sync, Import.sync, and Export.sync directories under server_vault are not backed up.

Note: The backup creates symbolic links to vault data if your Backup.sync area is on a different partition from your server or if your system does not support hardlinks. If the vault data is removed (for example, by an rmfolder command) the backed-up symbolic links cannot be used to restore your data. To avoid this problem, archive your Backup.sync area using a switch to de-reference the symbolic links.

The back-up operation does not make the server completely inaccessible. The server is suspended for a short time at the beginning of the back-up but after that it is accessible for all ProjectSync operations. DesignSync users have read access to operations after the initial suspension. For example, after the initial suspension, DesignSync users can populate from a vault during a back-up.

For each dated back-up subdirectory, a .cleanup executable file is generated in the Backup.sync directory. You can run this executable

```
to remove the corresponding backed-up data. For example, if your
back-up directory is:

  $SYNC_DIR/../syncdata/<host>/<port>/server_vault/Backup.sync/
  20010710_145601

You would enter "20010710_145601.cleanup" to remove the data in
the 20010710_145601 directory.

If the clean-up operation fails, you get an error message.

During the backup, .inprogress is appended to the dated
subdirectory of Backup.sync. This extension is removed when the
back-up completes. You can use this feature to check the status of
your back-up. If .inprogress is appended to the subdirectory
name and you cannot write to the server, the back-up is still
underway. If .inprogress is appended to the subdirectory name
and you can write to the server, the server crashed and
restarted while the back-up was underway.

See the ProjectSync User's Guide: "Backing Up Your Server" help topic
for information on recovering from server crashes, restoring
your backed-up files, and enabling and scheduling automated backups.

Note: If you choose to back up without using the backup command,
you must stop your server, perform the backup, and then restart
the server. When you use the backup command, all access to the
metadata is refused while the metadata is copied; all write access
to the vault is refused while the vault is copied. The metadata
and the vault are backed up into a single archive. Most of the
$SYNC_CUSTOM_DIR is not included in the back-up operation.
```

SYNOPSIS

```
backup [-from <date>]
```

OPTIONS

- -from

-from

```
-from <date>          When the -from argument is specified, an
                      incremental backup is performed based on
                      the backup that occurred on <date>. The
                      <date> value must match the timestamp of
                      a previous full or incremental backup
                      directory. For example, if you have a
                      backup directory called "20031027_085512"
                      then you can specify:
```

```
backup -from "20031027 085512"

You also can specify the special value
"last" to incrementally back up from
the most recent back up.
```

RETURN VALUE

```
Name of the back-up directory
```

SEE ALSO

```
rstcl, access verify
```

EXAMPLES

```
The following example illustrates how to use the backup command in
a Tcl script. The script first calls access verify to ensure that
the user has permission to perform a back-up. If not, the script
returns "Permission denied." If the user has permission, the back-
up operation performs an incremental backup based on the previous
backup. If the operation fails, the script issues an error.

  if {[access verify AdministrateServer $SYNC_User]} {
       if {[catch { backup -from last } result]} {
           puts ""
           puts "** Backup Server Error **"
           puts "Stack Trace: $errorInfo"
           puts ""
       }
  } else {
           puts "Permission denied."
  }

When using backup in a script, invoke the access verify command to
ensure that only authorized users can back up the server. See the
ENOVIA Synchronicity Access Control Guide for information on using
the AdministrateServer access control to restrict access to server
operations.
```

keydbcheckpoint

keydbcheckpoint Command

NAME

```
keydbcheckpoint     - Back up for module-related metadata tables
```

DESCRIPTION

```
This command is used internally to maintain the integrity of the
module-related metadata tables.
```

SYNOPSIS

```
keydbcheckpoint [-restore] [--]
```

restoreserver

restoreserver

NAME

```
restoreserver       - Restores the data from a backed-up server
```

DESCRIPTION

```
The restoreserver script restores an entire server from your
backed-up server data. This script lets you restore not only vault
folders but also metadata, notes, and attachments. When you restore a
Cadence view, all the objects in the view are restored. (Note: When
you restore a collection object other than a Cadence view object, you
must restore both the main collection object and all of its member
file objects.)

Your SyncServer is shut down during the restoration and restarted again
when the operation is complete. You must be the server owner to run the
restoreserver script.

Before running the restoreserver script, you need to restore the
server_vault and $SYNC_CUSTOM_DIR data from your system backup and then
transfer the data in the Backup.sync area to the correct place within
the server.

Run the restoreserver script from the command line on the system where
you run your SyncServer(s). To invoke the script, enter the following
on the command line:

   restoreserver

The script starts and opens a log file, restoreserver.log, in your
home directory.
```

If your SyncServer has more than one port, the script prompts you to choose the port you want to restore. Enter the number for the port.

If you have more than one set of backup data, the script prompts you to choose the backup data that you want to restore from. The script displays a numbered list showing each dated incremental and full backup. Enter the number for the date you want to restore from.

If you choose an incremental backup date, the script first restores the last full backup and then restores each subsequent incremental backup.

See "Restoring All Server Backup Data" in the ENOVIA Synchronicity DesignSync Data Manager Administrator's Guide for complete details on restoring your server data.

SYNOPSIS

restoreserver

RETURN VALUE

The restoreserver command logs its activity to the restoreserver.log file in your home directory. If the command fails, DesignSync returns an error explaining the failure.

SEE ALSO

Backup

restorevault

restorevault Command

NAME

restorevault - Restores specified server vault or directory data

DESCRIPTION

This command lets you specify backed-up vault data to restore. You can restore vault folders, directories, or individual vault objects. When you restore a Cadence Cell View collection object, all vault objects within the equivalent view folder are also

```
restored. However, to restore vault data of other types of collection
objects, you must restore both the collection object and each member
file object.

This command does not restore:
 o  Backed-up metadata, notes, or attachments. For a full
    restoration of these objects, you must use the restoreserver
    script. See Restoring All Server Backup Data in the ProjectSync
    User's Guide for details.
 o  Data for the Hierarchical Configuration Manager.
 o  Vaults backed up from pre-4.0 versions of Developer Suite.

The restorevault command is server-side only and must be run
using the rstcl command or by passing a script in a URL from
your browser. See the 'server-side' topic or the ENOVIA Synchronicity
stcl Programmer's Guide for details.

If you incorporate the restorevault command into an rstcl script,
the access controls that govern who can restore a vault are
not applied.

Error messages are written to error_log in:
$SYNC_DIR/custom/servers/<server>/<port>/logs
```

SYNOPSIS

```
restorevault -from <backup_area> [-overwrite] [--] <path>
```

OPTIONS

- -from
- -overwrite
- --

-from

```
-from <backup_area>  The name of the directory inside Backup.sync
                     that you want to restore from. For example,
                     20030813_095027.
```

-overwrite

```
-overwrite           Specifies that items that exist in the vault
                     are overwritten during the restoration. (See
                     Restoring Your Server Vault Data in the
                     ProjectSync User's Guide for examples of using
                    this option.)
```

--

```
--                  Indicates that the command should stop
                    looking for command options. Use this
                    option when the object you specify begins
                    with a hyphen (-).
```

OPERANDS

- Vault Directory Path

Vault Directory Path

```
<path>              The path of the vault directory or vault
                    object to be restored. If you specify a
                    directory, the restoration is recursive.
```

RETURN VALUE

```
None
```

SEE ALSO

```
backup
```

EXAMPLES

```
This example tcl script uses the restorevault command to restore
the vault data from the and4 directory. Any existing files in the
directory are overwritten:

restorevault -overwrite -from 20030707_150817 /Projects/smallLib/and4

This example restores vault data of the Cadence view object called
layout and all of its member objects.

restorevault -from /Projects/smallLib/mid2/layout.sync.cds
```

suspend

suspend Command

NAME

```
suspend             - Sets the server to a semi-active state
```

DESCRIPTION

```
The suspend command sets the server to a semi-active state.  This
command should only be used inside a server-side TCL script. The
script should be invoked through rstcl.  It will execute the suspend
command, run the code specified in the tcl-code argument and terminate
the suspend, restoring the server to normal operation.

While the server is in a semi-active state, DesignSync rejects
operations such as checkin, with a message explaining that the server
is in a suspended state and returning the optional -because message.
The administrator can define registry settings that determine whether
the operation will retry, how many retry attempts will be attempted
before the command fails, and how long to wait between retries.  For
more information, see the ENOVIA DesignSync Data Manager
Administrator's Guide.

Note: The backup functionality in DesignSync uses the suspend command
to put the server in a restricted access state.
```

SYNOPSIS

```
suspend [-because <why>] -maintenance | -readonly [-mode]
        {tcl-code}
```

ARGUMENTS

- tcl Code

tcl Code

```
<tcl-code>      The code to execute while the server is in suspend
                mode.
```

OPTIONS

- -because
- -maintenance
- -mode
- -readonly

-because

```
-because <why>  Specifies the reason the server is in semi-active
```

state. Any operations that fail to run while the server is in this state will return this as a reason to the user.

-maintenance

-maintenance Sets the server to deny all read or write operations.

This option is mutually exclusive with -readonly. You must specify either -readonly or -maintenance.

-mode

-mode Sets a return string indicating whether the server is in 'normal,' 'readonly,' or 'maintenance,' mode. This can be used by scripts wishing to check the server status.

-readonly

-readonly Sets the server to allow read-only vault operations, such as populate -get/-share, or compare, contents, ls, and other read-only commands. It does not allow operations that modify the vault such as populate -lock, ci, mkmod, etc.

This option is mutually exclusive with -maintenance. You must specify either -readonly or -maintenance.

RETURN VALUE

Not applicable.

SEE ALSO

backup, rstcl

EXAMPLES

This example shows a procedure that releases the global server lock, suspends the server for 100 seconds in both maintenance and readonly

```
mode.

Releasing the global server lock is recommended to allow other
commands past the locking gate to receive the 'in maintenance mode'
failure.  Otherwise, the lock prevents other server-side tcl scripts
from being processed by the server.

# first release the global lock
# To avoid any output from the 'url syslock' command,
# the record command places the output in the variable "msg"

record {url syslock -release smdSrvrMetaDataLock} msg

proc FullMaintenanceCode {} {
    # here the code that requires the server to reject
    # all requests, will be done
    after 100000
}

proc ReadOnlyMaintenanceCode {} {
    # here the maintenance which allows read ops
    # such as (populate -get/-share, ls, compare...)
    # but denies write ops,
    # such as (ci, populate -lock, tag, rmversion....)
    after 100000
}

suspend -maintenance \
-because "Server Is Undergoing Required Maintenance" \
{FullMaintenanceCode}

suspend -readonly \
-because "Server Is Undergoing Required Maintenance. \
Read Operations are allowed" \
{ReadOnlyMaintenanceCode}
```

Troubleshooting

syncinfo

syncinfo Command

NAME

```
syncinfo            - Returns Synchronicity environment information
```

DESCRIPTION

```
This command returns information about the Synchronicity
software environment, such as version number, location of
registry files, and default editor and HTML browser. The command
```

```
can be run from the client to return client information, or from
the server to return server information.

By default (with no arguments specified), all available information
is returned. You can request specific information by specifying
one or more command arguments.

If a given value has not been set or is not available, then
'syncinfo' returns an empty string. For example, if you ask for
portRegistryFile from the client, the return value is empty because
portRegistryFile is only available from the server.
```

SYNOPSIS

```
syncinfo [<arg> [<arg>...]]
```

ARGUMENTS

General Information

isServer

```
isServer            Returns a Tcl boolean value (0 or 1) indicating
                    whether the software executing the syncinfo
                    command is acting as a server (1) or client (0).
```

syncDir

```
syncDir             Returns the root directory of the Synchronicity
                    software installation. On UNIX, this value
                    corresponds to the SYNC_DIR environment
                    variable (on Windows, SYNC_DIR is not required).
```

version

```
version             Returns the version of the Synchronicity software
                    as a string.
```

Registry Information

clientRegistryFiles

```
clientRegistryFiles    Returns a comma-separated list of registry
                       files used by the Synchronicity clients
                       (DesSync, stcl, dss, stclc, dssc).
```

enterpriseRegistryFile

```
enterpriseRegistryFile Returns the enterprise-wide registry file.
```

portRegistryFile

```
portRegistryFile       Returns the port-specific registry file.
```

projectRegistryFile

```
projectRegistryFile    Returns the project-specific registry file.
```

serverRegistryFiles

```
serverRegistryFiles    Returns a comma-separated list of registry
                       files used by a Synchronicity server.
```

siteRegistryFile

```
siteRegistryFile       Returns the site-wide registry file.
```

syncRegistryFile

```
syncRegistryFile       Returns the Synchronicity-supplied standard
                       registry file.
```

userRegistryFile

```
userRegistryFile       Returns the user-specific registry file.
```

usingSyncRegistry

```
usingSyncRegistry      Returns a Tcl boolean value (0 or 1)
                       indicating whether the Synchronicity
                       software is using the text-based registry (1)
                       or the native Windows registry (0).
```

Customization Information

customDir

customDir	Returns the root directory of the 'custom' branch of the Synchronicity installation hierarchy, which contains all site- and server-specific customization files. The default value, <SYNC_DIR>/custom, can be overridden by the SYNC_CUSTOM_DIR environment variable.

customSiteDir

customSiteDir	Returns the directory that contains site-specific customization files. The default value, <SYNC_CUSTOM_DIR>/site (which defaults to <SYNC_DIR>/custom/site), can be overridden by the SYNC_SITE_CUSTOM environment variable.

customEntDir

customEntDir	Returns the directory that contains enterprise-specific configuration files. The default value, <SYNC_ENT_CUSTOM> (which defaults to <SYNC_CUSTOM_DIR>/enterprise), can be overridden by the SYNC_ENT_CUSTOM environment variable.

siteConfigDir

siteConfigDir	Returns the directory that contains site-specific configuration files. The default value, <SYNC_SITE_CUSTOM>/config (which defaults to <SYNC_CUSTOM_DIR>/site/config, which defaults to <SYNC_DIR>/custom/site/config), can be overridden by the SYNC_SITE_CNFG_DIR environment variable.

usrConfigDir

userConfigDir	Returns the directory that contains user configuration files. The default value, <HOME>/.synchronicity, can be overridden by the SYNC_USER_CFGDIR environment variable.

userConfigFile

userConfigFile	Returns the user configuration file. The default value, <HOME>/.synchronicity/user.cfg, can be overridden by the SYNC_USER_CONFIG

environment variable.

Client Information

connectTimeout

connectTimeout	Returns the number of seconds the client will wait per communication attempt with the server.

commAttempts

commAttempts	Returns the number of times client/server communication is attempted before failing. Using multiple attempts protects against transient network problems. 'Connect Failure' failures do not trigger multiple connection attempts, because transient network problems rarely cause this error. Note: When the number of communication attempts is the default value of 3, 'syncinfo commAttempts' returns no value instead of returning 3.

defaultCache

defaultCache	Returns the default cache directory for the client as specified during installation or using SyncAdmin.

fileEditor

fileEditor	Returns the default file editor as specified during installation or using SyncAdmin.

htmlBrowser

htmlBrowser	(UNIX only) Returns the default HTML browser as specified during installation or using SyncAdmin.

proxyNamePort

proxyNamePort	Returns the <name>:<port> of a proxy, if one is defined in a client registry file or

using the ProxyNamePort environment variable.

somTimeout

somTimeout	Returns the number of milliseconds after an unsuccessful server connection attempt during which the client does not try to connect again. This timeout protects against an operation on many objects (such as 'ls' on a large directory) taking an excessively long time to complete when there is a connection failure (such as when the server is down). Instead of waiting the connectTimeout period for each object, the operation fails for all objects after the first connection failure.

Server Information

berkdbIsShmEnabled

berkdbIsShmEnabled	For Synchronicity internal use only.

berkdbShmKey

berkdbShmKey	For Synchronicity internal use only.

isTestMode

isTestMode	For Synchronicity internal use only. Returns a Tcl boolean value (0 or 1) indicating whether the software executing the syncinfo command is running in test mode (1) or not (0). This feature is useful for regression testing of servers.

serverMetadataDir

serverMetadataDir	Returns the directory that contains the server metadata (such as relational database) files.

serverDataDir

```
 serverDataDir       Returns the directory that contains vault
                     (repository) data that is stored by a server.
```

serverMachine

```
 serverMachine       Returns the name of the server as returned by
                     gethostname(). This value is returned only
                     when 'syncinfo' is run from a server-side script.
```

serverName

```
 serverName          Returns the name of the server as it was
                     specified in the URL used to contact the
                     server. This value is returned only when
                     'syncinfo' is run from a server-side script.
```

serverPort

```
 serverPort          Returns the port number used by the server to
                     respond to the syncinfo request. This value is
                     returned only when 'syncinfo' is run from a
                     server-side script.
```

User Information

home

```
 home                Returns the home directory of the user
                     running syncinfo (HOME on UNIX, or as
                     defined in your user profile on Windows platforms).
```

userName

```
 userName            Returns the account name of the user
                     running syncinfo.
```

RETURN VALUE

```
In dss/dssc mode, you cannot operate on return values, so the return
value is irrelevant.

In stcl/stclc mode:
  - If no argument is specified, the return value is a
```

```
    name/value list (Tcl 'array get' format) containing
    all available information.
  - If a single argument is specified, the return value is
    the requested value (not a list).
  - If more than one argument is specified, the return value
    is a name/value list containing the requested information.
  - If any argument is not known, an exception is thrown.
```

SEE ALSO

```
server-side
```

EXAMPLES

- Example Showing the SyncInfo Version on Client Startup
- Example of Extracting SyncInfo Information to an Array
- Example Showing Extracting the Information from an Array
- Example of extracting Name/Value Pairs for Specific Arguments

Example Showing the SyncInfo Version on Client Startup

```
When you start any Synchronicity client, 'syncinfo version'
executes, which displays (and writes to your log file
if logging is enabled) the Synchronicity version. In this
example, the software is version 3.0.
  % stclc
  Logging to c:\goss\dss_01192000_092559.log
  V3.0

  stcl>
```

Example of Extracting SyncInfo Information to an Array

```
The following stcl script fragment shows how to get all known
information as a Tcl array variable. The 'version' string is
then printed.
  array set info [syncinfo]
  puts "Version: $info(version)"
```

Example Showing Extracting the Information from an Array

```
This example uses the single-argument form of syncinfo to print the
same version information provided by the previous example:

  puts "Version: [syncinfo version]"
```

Example of extracting Name/Value Pairs for Specific Arguments

```
The following example uses command arguments to return a list
of the 'syncDir' and 'userName' values. This example
also shows how to enumerate the name/value list returned by
syncinfo without storing it in an array variable.
  foreach {name value} [syncinfo syncDir userName] {
      puts "$name: $value"
  }
```

synctrace

synctrace Commands

NAME

```
synctrace           - Commands to help diagnose software problems
```

DESCRIPTION

```
The 'synctrace' commands help Synchronicity diagnose software
problems and performance issues by enabling or disabling software
tracing.

See the "synctrace set" command for details. Also see "Running a
DesignSync Client in Debug Mode" in DesignSync Data Manager User's
Guide.
```

SYNOPSIS

```
synctrace [un]set [-server <serverURL>] 0
```

SEE ALSO

```
synctrace set, synctrace unset
```

EXAMPLES

```
See the "synctrace set" command.
```

synctrace set

synctrace set Command

NAME

```
synctrace set       - Turns on software tracing
```

DESCRIPTION

```
Helps Synchronicity diagnose software problems and performance
issues by enabling software tracing. Synchronicity may ask you to
enable tracing to help with problem diagnosis.

When you use the synctrace command to set the trace level for a
client or server, the trace is only in effect for the current
client or server session. The trace terminates when you shut down
the client or the server and does not start again when you restart
the client or server, unless you reinvoke the command. If you have
already set (or unset) synctrace, and you inadvertently set (or
unset) the trace again, the second setting has no effect.

The trace output is stored in the
<SYNC_USER_CFGDIR>/logs/sync_client_trace_<date>_<time>.log file.
In addition, the output is stored in the dss_<date>_<time>.log
file. The trace output for a server is stored in
<SYNC_CUSTOM_DIR>/servers/<host>/<port>/logs/error_log.

If you want to have tracing on when you start the client or server,
you should set the SYNC_TRACE environment variable to 0.
See "Running a DesignSync Client in Debug Mode" in DesignSync Data
Manager User's Guide
for more details.
```

SYNOPSIS

```
synctrace set [-server <serverURL>] 0
```

OPTIONS

- -server

-server

```
-server <serverURL>    Turns on the trace for the server you specify.
                       If you omit the -server switch, the trace is
                       turned on for the client session from which
                       you invoked the command. Specify the URL
                       as follows:
```

```
     sync://<host>[:<port>]
Where 'sync://' is required, <host> is the
machine on which the SyncServer is
installed, and <port> is the SyncServer port
number (if omitted, defaults to 2647/2679). For
example, you might specify the following:
-server sync://serv1.abco.com:30138

Note: DesignSync also supports a syncs
protocol for communicating with secure (SSL)
SyncServer ports. In most cases, DesignSync
automatically redirects requests to a
cleartext (non-secure) port using the sync
protocol to the secure port, if one is
defined. The default Synchronicity secure
port number is 2679. Your Synchronicity
administrator defines what SyncServer ports
are available and whether secure
communications are required. See the
"Overview of Secure Communications section"
in DesignSync Data Manager User's Guide for
more information.

Enables tracing on all software
components. While it is possible to
selectively turn on tracing for specific
software libraries, it is typically most
useful to turn on all tracing.
```

RETURN VALUE

```
none
```

EXAMPLES

- Example of Turning Tracing on for all Libraries
- Example of Turning Trace off for All Libraries
- Example of Turning Trace On for a Specific Server
- Example of Turning Trace Off for a Specific Server

Example of Turning Tracing on for all Libraries

```
The following example turns on tracing for all libraries.
  stcl> synctrace set 0
```

Example of Turning Trace off for All Libraries

```
The following example turns off tracing for all libraries.
```

```
stcl> synctrace unset 0
```

Example of Turning Trace On for a Specific Server

```
The following example turns on tracing for the specified server:
  stcl> synctrace set -server sync://serv1.abco.com:30138 0
```

Example of Turning Trace Off for a Specific Server

```
The following example turns off tracing for the specified server:
  stcl> synctrace unset -server sync://serv1.abco.com:30138 0
```

synctrace unset

synctrace unset Command

NAME

```
synctrace unset     - Turns off software tracing
```

DESCRIPTION

```
See the "synctrace set" command.
```

SYNOPSIS

```
synctrace unset [-server <serverURL>] 0
```

Utilities

convertdata

convertdata

NAME

```
convertdata         - Converts CVS/RCS vault files to DesignSync format
```

DESCRIPTION

The convertdata utility converts vault files from Concurrent Version System (CVS) or Revision Control System (RCS) data formats to the DesignSync format. See the DesignSync Data Manager Administrator's Guide: "Converting a Vault Repository from CVS/RCS Format to DesignSync Format".

convertutil

convertutil

NAME

convertutil - Converts between supported vault file formats

DESCRIPTION

The convertutil utility lets you recursively convert a vault folder from one format to another. The convertvault utility, another utility for converting vaults, provides more flexibility than the convertutil utility; however, for most conversion tasks, convertutil is sufficient. See the DesignSync Data Manager Administrator's Guide: "Converting Vault Data".

convertvault

convertvault

NAME

convertvault - Converts specified vault files to supported formats

DESCRIPTION

The convertvault utility lets you convert a single vault file type to another supported type or you can convert multiple vault files at one time. The convertutil utility, another utility for converting vaults, provides less flexibility than the convertvault utility; however, for most conversion tasks, convertutil is sufficient. See the DesignSync Data Manager Administrator's Guide: "Converting Vault Data".

exportVaults

exportVaults

NAME

```
exportVaults        - Exports DesignSync vault folders
```

DESCRIPTION

```
Use the exportVaults utility to export data from client vault folders.
See DesignSync Data Manager User's Guide: "Using the Vault
Utilities." For other export scenarios, use the ProjectSync Export
Projects feature. See ProjectSync User's Guide:"Exporting Projects."
```

importVaults

importVaults

NAME

```
importVaults        - Imports DesignSync vault folders
```

DESCRIPTION

```
Use the importVaults utility to import data converted using the
convertdata utility or data exported from a client vault. See
DesignSync Data Manager User's Guide: "Using the Vault Utilities."
For other import scenarios, use the ProjectSync Import Projects
feature. See ProjectSync User's Guide: "Importing Projects."
```

SyncAdmin

SyncAdmin

NAME

```
SyncAdmin           - Synchronicity Administrator tool
```

DESCRIPTION

```
Synchronicity's SyncAdmin tool is a graphical user interface that
lets system administrators, project leaders, and users configure
DesignSync clients (command-line and graphical) for
site, project, or individual use.

You execute SyncAdmin from your operating system shell, not
```

```
from a Synchronicity client shell (dss/dssc/stcl/stclc).  On
Windows platforms, you invoke SyncAdmin from the Windows
Start menu, typically:
  Start->Programs->Dassault Systems <version>->SyncAdmin

See SyncAdmin help for details on SyncAdmin. From the GUI, click
the Help button on any SyncAdmin page.
```

SYNOPSIS

```
SyncAdmin [-file <filename> | -project | -site | -user]
```

OPTIONS

-file

```
-file <filename>  Edit the specified registry file, bypassing the initial
                  SyncAdmin window (where you select which registry
                  file to edit).
```

-project

```
-project          Edit the project registry file, bypassing the initial
                  SyncAdmin window (where you select which registry
                  file to edit).
```

-site

```
-site             Edit the site registry file, bypassing the initial
                  SyncAdmin window (where you select which registry
                  file to edit).
```

-user

```
-user             Edit the user registry file, bypassing the initial
                  SyncAdmin window (where you select which registry
                  file to edit).
```

RETURN VALUE

```
none
```

SEE ALSO

```
DesSync
```

EXAMPLES

```
This example invokes SyncAdmin:
  % SyncAdmin

This example invokes SyncAdmin, in background mode, to edit the
user registry:
  % SyncAdmin -user &
```

syncdadmin

syncdadmin Command

NAME

```
syncdadmin          - Manages syncd processes
```

DESCRIPTION

```
This command manages Synchronicity daemon (syncd) processes on a
per-user basis. 'syncdadmin' is not a DesignSync shell command, but
is instead a standalone utility that you invoke from a standard
shell or a shell script.

The syncd process manages communication between dss/stcl
sessions and SyncServers. The syncd process can manage multiple
dss/stcl requests per user, allowing one user to run parallel
dss/stcl sessions. Note that dssc and stclc do not use syncd;
they communicate directly with a SyncServer.

When you invoke dss or stcl, they connect to your syncd process if
one is already running. Otherwise, they attempt to start
syncd. There is at most one syncd process running per user per
machine at any time.

On Unix systems, the syncd process times out after 180 minutes of
inactivity (after the last dss/stcl session communicating with the
```

```
syncd process exits). The syncd process will not time out if there
is an active dss or stcl session, or if there is a lock on the
syncd process. You can define the SYNC_DAEMON_TIMEOUT environment
variable to change the default time-out period of 180 minutes:

         setenv SYNC_DAEMON_TIMEOUT <n>

where <n> is the number of minutes syncd waits before timing out.

On Windows platforms, syncd never times out. However, you can
stop syncd from the Windows Start menu, typically:
  Start->Programs->Dassault Systems <version>->Stop SyncDaemon

Note: When using dss/stcl, many environment changes (including setting
SYNC_DAEMON_TIMEOUT) do not take effect until syncd is stopped and
restarted.
```

SYNOPSIS

```
syncdadmin [begin | close [-force] | lock | start |
            status [-verbose] | stop [-force] | unlock |]
```

ARGUMENTS

- begin
- close
- lock
- start
- status
- stop
- unlock

begin

```
begin       Same as 'start'. The 'begin' argument may be removed in a
            future release.
```

close

```
close       Same as 'stop'. The 'close' argument may be removed in a
            future release.
```

lock

```
lock        Locks the syncd process so that 'syncdadmin stop'
            will not terminate syncd (unless you use the -force
```

```
option). 'syncdadmin lock' also starts syncd if it
is not already running.

You typically lock syncd at the beginning of a shell
script that calls dss/stcl several times. The lock
prevents syncd from being terminated inadvertently.
For example:

    #!/bin/csh
    syncdadmin lock
    dss ci *.cpp
    <some shell (non dss/stcl) commands>
    dss ci *.h
    syncdadmin unlock
```

start

```
start       Starts the syncd process if one is not already running.
            The 'syncdadmin start' command is typically used in a
            shell script to invoke syncd (if necessary) before
            invoking Synchronicity commands. For example:

              #!/bin/csh
              syncdadmin start
              dss ci ...

            In this script fragment, 'syncdadmin start':
              - Starts syncd only if syncd is not already running
              - Does not allow the next line of the script to
                execute until syncd is running

            Note: There is a short start-up time associated with syncd,
                  during which it does not accept commands. To prevent
                  a race condition, avoid starting syncd as follows:

                  % syncd &
                  % dss ci ...    << Likely to fail because syncd is
                                     still starting up
```

status

```
status      Indicates whether the syncd process is currently
            running and on which machine. Use the -verbose
            option to output additional information, such as
            whether syncd is locked.
```

stop

```
stop        Attempts to terminate the syncd process. The syncd process
            will not terminate if there is an active dss/stcl session
```

```
            running or if a lock has been set on the syncd process.
            Use the -force option to override these constraints.
```

unlock

```
 unlock     Removes the lock from a syncd process. You typically
            lock syncd from a script that calls dss/stcl multiple
            times. You would then unlock syncd at the completion
            of the script.
```

RETURN VALUE

```
 none
```

EXAMPLES

```
 This example displays the syncd status before and after starting
 syncd:
   % syncdadmin status
   SYNC: syncd is not running.
   % syncdadmin start
   SINC: Attempting to spawn daemon.
   SYNC: Syncd ready.
   % syncdadmin status
   SYNC: syncd is running on linus.
   %
```

sync_setup

sync_setup Command

NAME

```
 sync_setup           - Installing and configuring the DesignSync server
```

DESCRIPTION

```
 This script installs or updates the configuration of the DesignSync
 server. You can run this script interactively, to set up a single
 server, or call it from a script to perform a non-interactive
 DesignSync server setup on a single service or multiple servers at
 once.

 Note:  Passwords should ideally not be passed through the the script,
```

```
but specified as a key/value pair on the command line, particularly
with respect the postgres database password.  The ENOVIA 3DPassport
password can be encrypted and passed, encrypted to the script, but
the postgres must be in cleartext.  Therefore it is not necessarily
desirable to include it in any scripts or in the XML configuration
file.

Tip: If you want to automatically start any servers that have been
installed/configured by the sync_install script, do any of the
following:
 o Set the StartDesignSyncServer setting to true in the SiteSettings
   section of the configuration file, to start all the servers
   installed/configure during the command operation.
 o Set the StartDesignSyncServer setting to true in the
   ServerSettings section of the configuration file for all servers you
   want to start.
 o Use the return value of the command, 0, indicating success, to
   start all the servers configured by the command.

 DesignSync recommends using the StartDesignSyncServer setting rather
 than the command return value.
```

SYNOPSIS

```
sync_setup [-config=<UserConfig.xml>] [-[no]debug] [-dryrun]
           [-log=<filename>]
           [-report={normal|verbose}] [<key>=<value> [...]]
```

ARGUMENTS

- key/value pair

key/value pair

```
<key>=<value>     Optional arguments to specify to the sync_setup
                  script.  Any options specified to the script allow
                  become the default option for any settings that
                  hat use that information, or ovrride any settings
                  set in the file specified by the -config switch,
                  any environment variables, or the defined system
                  defaults.
```

OPTIONS

- -config
- -[no]debug
- -[no]dryrun
- -log
- -report

-config

-config= <UserConfig.xml>	The name of the XML file containing the the list of defined settings for the sync_setup script to use. All the settings can be defined within a single script, even if you are configuring multiple servers. For information on defining the user script and a sample script, see the ENOVIA Synchronicity DesignSync Administrator's Guide. Tip: If you want to use multiple configuration files within a batch operation, you can create a script that runs sync_setup more than once, each time specifying the desired configuration file. When -config is specified, the server does not allow any input from the user. If any options are missing, the command fails with an appropriate error message.

-[no]debug

-[no]debug	This option indicates whether debugging is enabled. This helps the Dassault Systems support team diagnose server installation problems by enabling tracing during the server installation and configuration process. Use this option only if you are experiencing problems with software installation. -debug turns on tracing for the sync_setup script. This should be used only when sync_setup is failing and you need help understanding why. -nodebug indicates that debug information should not be output by the command. (Default) Tip: When you use this option, you should specify a log file for the output so it can be reviewed by the support team.

-[no]dryrun

-[no]dryrun	Specifies whether to treat the operation as a trial run or perform the server configuration. -dryrun does not perform server configuration. Using the -dryrun option helps detect problems that might prevent the server

installation/configuration operation from succeeding. The dryrun can show you which settings are specified incorrectly, or need to be specified, but are not in the configuration file.
Errors such as permissions or inaccessible servers are not reported by a dry run. Note that a dry run setup is faster than a normal server installation/configuration. If -dryrun is specified, no servers are are actually configured.

-nodryrun perform the server configuration/installation as specified. (Default)

-log

-log <filename> Specify the name of the log file. If the filename doesn't exist, DesignSync creates it. If the file does exist, DesignSync appends the new information to the end of the log file.

The filename can be specified with an absolute or relative path. If you specify a path for the log file, the directory you specify must already exist and you must have write permissions to the directory in order for the log to be placed into it, DesignSync does not create the path.

-report

-report [error|brief|normal verbose] Controls the verbosity of the output to the screen or to the log file.

-report error reports only the errors.
-report brief reports only the errors.
-report normal reports the typical messages seen when sync_setup is run interactively.
-report verbose reports all of the generated output.

RETURN VALUE

Returns a value of 0 is the command runs successfully. If the command is unsuccessful, you receive a message explaining the failure.

SEE ALSO

```
synctrace
```

EXAMPLES

ProjectSync Data Manipulation

Note Manipulation

note

note Commands

NAME

```
note                - Server-side commands for accessing notes
```

DESCRIPTION

```
Users create notes and manage note links using the ProjectSync
graphical interface. The 'note' commands are for advanced users who
need programmatic access to note capabilities.

The 'note' family of commands provide access to the note web
object type. URLs for notes have the following form:

   sync:///Note/SyncNotes/<notetype>/<noteid>

For example, the following URL specifies the 5th note of the
BugReport notetype:

   sync:///Note/SyncNotes/BugReport/5

Notes can only be accessed from server-side scripts, so always use
the sync:/// syntax (no <host>:<port> specification).

Note: The "notetype schema" command provides access to the database
structure of notetypes.
```

SYNOPSIS

```
note <note_command> [<note_command_options>]

Usage: note [attach|counts|create|delete|detach|getprop|links|query|
             relink|setprops|systems|]
```

EXAMPLES

```
See specific "note" commands.
```

note attach

note attach Command

NAME

```
note attach          - Creates a link between a note and an object
```

DESCRIPTION

```
This command attaches a note to the specified object, creating
a notelink -- a link between an object and a note.

The <noteURL> argument must be a reference to an existing note.
The <objURL> argument can be a reference to any valid object,
including another note. If <objURL> is a local note URL, then
the note that it refers to also must exist. However, for any
other <objURL> type, including notes on remote servers, the
existence of the object is not checked. Only the structure of
the URL is checked.

It is not possible to create a duplicate notelink. However, the
attempt does not cause an error; attempts to create duplicate
 notelinks are silently ignored.

The successful execution of this command generates an atomic
note attach event and fires the corresponding triggers
in response.

This command is available only from server-side scripts.
```

SYNOPSIS

```
note attach <noteURL> <objURL>
```

OPTIONS

```
none
```

RETURN VALUE

```
none
```

SEE ALSO

```
note detach, note links, server-side, rstcl
```

EXAMPLES

- Example of Attaching a Note to a Project
- Example of Attaching a Note to a Tagged Configuration

Example of Attaching a Note to a Project

```
This example attaches a bug report to the Asic project:

  note attach sync:///Note/SyncNotes/BugReport/2 sync:///Projects/Asic
```

Example of Attaching a Note to a Tagged Configuration

```
This example attaches the bug report to the Rel1 configuration
of the Asic project:

  note attach sync:///Note/SyncNotes/BugReport/2 \
              sync:///Projects/Asic@Rel1
```

note counts

note counts Command

NAME

```
note counts          - Computes statistics about notes and
                       the frequency of values
```

DESCRIPTION

```
This command runs a query against a note type and breaks
down the results according to the values it finds in selected
properties of the notes matched by the query. The breakdown can
be zero, one-dimensional, or multi-dimensional. Dimensions of
breakdown are note properties selected in the query, with the
results of the query grouped by value in the selected properties
(dimensions).

The results of the query (that is, the counts of how many notes
fit the search criteria and had the same values in the selected
properties) is stored in an output variable whose name is given
```

to the command. The command treats the variable as an array into which to store the results. The array has indices of all combinations of values that were found for the selected properties in notes matching the query criteria. The mapped values at those indices are the number of notes that had that particular combination of values.

For one-dimensional breakdowns (that is, breakdowns by a single property), the indices in the output array variable are the same as values found for that property in notes that matched the query. For two- or three-dimensional queries, the indices are a concatenation of values for each of the selected properties, separated by commas. If any of the values contain comma characters, then the comma characters are replaced with periods, so that the comma characters retain their separator semantics. The output array contains only non-zero entries.

The note counts command can operate on (break down by) properties of any type. However, the command is practical only for operations on enumerable property types, such as state machines, choice types, user fields, and perhaps integers. Results of breakdowns by floating-point properties and wide strings are generally not useful, but such use is not disallowed. Imprudent use of this command can copy very large amounts of data from the database into memory (for example, if you were doing a breakdown by the Body property of a note type).

You can generate simple time-based statistics by doing a breakdown on any Date or Timestamp property of a note type. In the resulting array, the indices will be dates and the values will be how many notes had that date value in that field. The resolution of the buckets for time-based statistics is controlled with the -dateresolution option. This option allows for specifying a unit of time (years, months, days, etc.) used to indicate the granularity of the statistical buckets.

This command is available only from server-side scripts.

SYNOPSIS

```
note counts <NotetypeName>
             [-countlinks] [-dateresolution <Resolution>]
             [ [-dbquery  <dbase_expr>] | [-sqlquery <sql_expr>] ]
             [<OutVarName> [Dimension0 [Dimension1 [Dimension2]]]]
```

OPTIONS

- -countlinks
- -dateresolution
- -dbquery
- -sqlquery

-countlinks

-countlinks	This option is used only for RevisionControl notes. Use this option to count the number of objects in each RevisionControl note instead of counting only the individual notes. When using this option, you must specify <OutVarName>.

-dateresolution

-dateresolution <Resolution>	If one or more of the Dimension arguments references a property of type Date or Timestamp, specifies a resolution of the bucketing of notes in the output array. The resolution is specified in terms of date granularity. The set of valid values are: years, months, weeks, and days. The value of the resolution affects the values used to form the indices in the returned array. The format for the index values for a given resolution are: years yyyy months yyyy-mm weeks yyyy-Www days yyyy-mm-dd If -dateresolution is not given as an option, the command defaults to a resolution of days.

-dbquery

-dbquery <dbase_expr>	A valid dBase query, which is converted to an equivalent SQL expression, and used to query the database. Analogous to the -dbquery option to note query.

-sqlquery

-sqlquery <sql_expr>	Filters the set of notes that are counted by the note counts command. Analogous to the -sqlquery option to note query. -sqlquery and -dbquery are mutually exclusive.

OPERANDS

Notetype Name

<NotetypeName>	The name of an existing note type.

Out Var Name

<OutVarName>	The name of a Tcl array variable in which to place specific notes that match the query criteria. The indices of the returned array are comma-separated concatenations of the Dimension0...2 property values. The array values are the number of notes that match the array index.

Dimensions

Dimension0 Dimension1 Dimension2	Up to three arguments that specify the bucketing criteria for the results. The values must be the name of an existing property name on the note type.

RETURN VALUE

The total number of notes that match the query criteria.

SEE ALSO

note query

EXAMPLES

Example Showing Reporting Against Fields in the Notetype

Suppose you have a note type called BugReport, with a State field (type SyncState), a Priority field (type SyncPriority) field, and a Resp field (type SyncUserList), plus all the standard note fields. This note type is populated with notes as follows:

Id	Author	Resp	State	Priority
1	caroline	cara	closed	high
2	jack	cara	closed	high
3	bert	ron	fixed	high
4	bert	mark	fixed	medium
5	lindsey	mark	fixed	high
6	bert	mark	analyzed	high
7	lindsey	jason	analyzed	low
8	bert	jason	closed	high
9	bert	mark	open	medium
10	caroline	mark	open	stopper

To find out how many BugReport notes are currently in-process (in any state except "closed"), you would specify:

```
note counts BugReport -dbquery "State#'closed'"
```

which would return the answer: 7

To get a breakdown of whom the notes are assigned to, give two extra parameters: 1. The name of a Tcl variable into which the note counts command will store its results. 2. The name of the note property that you wish to do a breakdown of - in this case, Resp:

```
note counts BugReport -dbquery "State#'closed'" MyMap Resp
```

This command also returns the number 7 but stores the following data in the array variable MyMap:

```
MyMap(ron)   1
MyMap(mark)  5
MyMap(jason) 1
```

Use array names or array get to extract the data from the MyMap array. For example:

```
array names MyMap
```

would return

"ron mark jason"

and

```
array get MyMap
```

would return

"ron 1 mark 5 jason 1"

To find the BugReports assigned to each engineer and get a breakdown by priority, you pass one additional parameter: the name of the additional property to break down by - in this case, Priority:

```
note counts BugReport -dbquery "State#'closed'" MyMap Resp Priority
```

You do not need to specify any additional output parameters when you add additional dimensions to the report; the results all go into the single output array parameter (MyMap), which in this example would be filled as follows:

```
MyMap(ron,high)     1
MyMap(mark,medium)  2
MyMap(mark,high)    2
MyMap(mark,stopper) 1
MyMap(jason,low)    1
```

The array contains only non-zero entries. For instance, user jason is not assigned any high-priority BugReports, so the report does not include the entry:

```
MyMap(jason,high) 0
```

Thus MyMap could be termed a sparse matrix.

It is also possible to further break down data by a third dimension.

In most cases, you would not use this command to generate this type of report. Like all the other note commands, the note counts command is best used as a building block.

Example Showing Time-Based Reporting on NoteTypes

This example illustrates time-based reporting.

To chart the incoming rate for BugReports, with a breakdown by month, you would use the following command:

```
note counts BugReport MyMap DateCreate -dateresolution months
```

The resulting map might look like this:

```
MyMap(2002-01) 2

MyMap(2002-02) 3

MyMap(2002-04) 5
```

This result indicates that two BugReports were filed in January 2002, three in February, and five in April. Empty buckets are not included: no BugReports were filed in March so there is no MyMap(2002-03) entry. Getting resolution by week would be difficult. You would have to use %W formatting (Week-of-year, 0-52) and convert the resulting data to get back to human-readable dates.

```
EXAMPLE 3
The following command itemized the objects in RevisionControl notes
for each command and and for each user.

note counts RevisionControl -countlinks MyMap Command Author
parray MyMap

The output of the MyMap variable is:

MyMap(ci,)                  = 0
MyMap(ci,Administrator)     = 1
MyMap(ci,Debra)             = 4
MyMap(ci,George)            = 0
MyMap(ci,Loren)             = 6
MyMap(co lock,George)       = 2
MyMap(co lock,Harry)        = 0
MyMap(co lock,Debra)        = 7
MyMap(co lock,Loren)        = 0
```

note create

note create Command

NAME

```
note create         - Creates a new note
```

DESCRIPTION

```
This command creates a new note of a specified note type.

Certain default behaviors apply when a note is created:

   * Notes are assigned a unique ID number that is 1 greater
     than the largest note ID number created in the database for
     a note of this type. However, two note create calls
     in a row would not necessarily return ID numbers that were
     incremented by 1. Another call in a different process could
     have created a note of the same note type.

   * The default creation date of a note (i.e., its DateCreate
     property) is set to the current server time.

   * The Author field defaults to the user ID (user name) of the
     person executing the command.

All of these default behaviors can be overridden with the note create
command.

You can set any number of properties when a note is created. If an
invalid value is supplied for any property, the entire note creation
```

process fails. If a property is not assigned an explicit value in the command, the default value for the note type is used. It is not an error to omit values for properties that are required by the note type.

If a property name is specified multiple times with different values, the last value is stored.

The successful execution of this command causes an atomic note create event and fires the corresponding triggers in response.

This command is available only from server-side scripts.

SYNOPSIS

```
note create [-date <date>] [-id <id>] -type <type_name>
            [{<name> <value>} [...]]
```

ARGUMENTS

- Name/Value for Note Properties

Name/Value for Note Properties

{<name> <value>}	Additional properties of the note, expressed as a list of name/value pairs. The first element of the pair is a valid property name of the note type and the second is the property value, which must be legal for the property type. If a value for the Id property is specified, use it as if the -id option had been used. An AMBIGUOUS_ID error is thrown if -id is used as well as an Id property value. Similarly, if both the -date argument and the DateCreate property are specified, an AMBIGUOUS_DATE error is thrown. Note that the curly braces specify the Tcl list syntax; they do not indicate a required argument as is true in most syntax descriptions. The note create command currently accepts multiple name/value pair lists for backward compatibility; however, this form is deprecated and Synchronicity strongly recommends avoiding this form.

OPTIONS

-date

-date <date>	A date value representing the creation date of the note. The <date> value should be an ISO-8601 formatted date in UTC - for example: 2003-05-29 12:34:56 If the -date option is omitted, the note creation time is the current time, stored in UTC. Although the creation time is stored in UTC, the GUI displays the creation time in the client's local time.

-id

-id <id>	The unique ID number for the note. If this number is not unique, the note is not created and you get the DUPLICATE_ID error code. If the -id option is omitted, the next available ID number is used. Each note type maintains its own internal ID generator. If the explicit ID number is higher than the currently existing internal ID of the generator, then the generator is adjusted to this new maximum ID number. That is, the next calls to note create with the -id option omitted will generate a number that is higher than the explicitly given ID number in this call. This behavior prevents collisions between the automatically generated IDs and explicitly created ones. If two processes are creating notes of the same note type and one is using the -id option while the other is using the default assigned ID, one of the processes is likely to fail because of conflicting ID numbers. If the delta number that the generator has to skip over is very high (e.g., above 10,000), then the command execution time may be much higher than normal, as the process has to skip over this delta. This case also increases the likelihood of failing with conflicting ID numbers.

-type

```
-type <type_name>  The name of a note type (for example, Note or
                   "BugReport") for which a new note is created.

                   A note type name is limited to 24 characters and
                   can contain only letters, numbers, hyphens, and
                   underscores. Note type names cannot begin with
                   hyphens or numbers and cannot contain spaces or
                   special characters.
```

RETURN VALUE

```
The URL of the new note.
```

SEE ALSO

```
note delete, note attach, note detach, note links, server-side, rstcl
```

EXAMPLES

- Example Showing Creating of a New Note with a Specific ID.
- Example Showing Creating a Note Using the Default ID

Example Showing Creating of a New Note with a Specific ID.

```
This example creates a new note of the Note notetype.

  note create -type Note \
    -id 6674 \
    -date 2002-05-29 \
    {Title "Hello, World!"} {Body "Main portion of a note."} \
    {Author goss}
```

Example Showing Creating a Note Using the Default ID

```
This example creates a new BugReport using the next available ID
number and sets the creation time as the current time:

  note create -type "BugReport" {Title "Broken!"} \
    {Body "It broke."} {Author norm}
```

note delete

note delete Command

NAME

```
note delete          - Deletes a note and associated notelinks
```

DESCRIPTION

```
This command deletes a note and any associated notelinks --
the links between the note and any objects to which it is
attached. The note delete command also deletes any files
attached to the note in fileattach fields. The note to be
deleted must exist.

The successful execution of this command causes an atomic note
delete event and fires the corresponding triggers in response.

This command is available only from server-side scripts.
```

SYNOPSIS

```
note delete <NoteURL>
```

OPTIONS

```
none
```

OPERANDS

- Note URL

Note URL

```
<NoteURL>            A valid URL for the note to be deleted.
```

RETURN VALUE

```
none
```

SEE ALSO

```
note detach, note links, note create, server-side, rstcl
```

EXAMPLES

```
This example deletes BugReport note 2:

  note delete sync:///Note/SyncNotes/BugReport/2
```

note detach

note detach Command

NAME

```
note detach         - Deletes the link between a note and an object
```

DESCRIPTION

```
This command detaches a note from the specified object, deleting
the notelink -- the link between the object and the note.

The <NoteURL> argument must be to a note object. The <ObjURL>
argument can be to any legal URL, including another note.

It is not an error to attempt to remove a notelink that does not
exist. However, <NoteURL> and <ObjURL> both must exist if the object
is a note. It is not an error to remove a non-note object that does
not exist. (Although, as with all note commands, the URLs must be
well-formed.)

The successful execution of this command causes an atomic note
detach event and fires the corresponding triggers in response.

This command is available only from server-side scripts.
```

SYNOPSIS

```
note detach <NoteURL> <ObjURL>
```

OPTIONS

```
none
```

OPERANDS

Note URL

```
<NoteURL>            A valid note URL.
```

Object URL

```
<ObjURL>             A valid URL to an object to detach
                     from the note.
```

RETURN VALUE

```
none
```

SEE ALSO

```
note attach, note delete, note links, server-side, rstcl
```

EXAMPLES

Example of Detaching a Bug Report from a Project

```
This example detaches the bug report from the Asic project:

  note detach sync:///Note/SyncNotes/BugReport/2 sync:///Projects/Asic
```

Example of Detaching a Bug Report from a Tagged Configuration

```
This example detaches the bug report from the Rel1 configuration
of the Asic project:

  note detach sync:///Note/SyncNotes/BugReport/2 \
              sync:///Projects/Asic@Rel1
```

note getprop

note getprop Command

NAME

```
note getprop        - Retrieves a property of a note
```

DESCRIPTION

```
This command retrieves the value of a single property on a note,
such as might have been stored with note setprops, url setprop,
or note create. The note must exist and the property name must be
one of the property names contained in the note type of the note.

This command only works in server-side scripts.
```

SYNOPSIS

```
note getprop <NoteURL> <PropertyName>
```

OPTIONS

```
None
```

OPERANDS

- Note URL
- Property Name

Note URL

```
<NoteURL>           The URL of a note, which must exist.
```

Property Name

```
<PropertyName>      The name of a property on the note.
```

RETURN VALUE

```
Returns the property value, as a string.
```

SEE ALSO

```
note setprops, url properties
```

EXAMPLES

```
This example extracts and prints the Title property of $noteURL,
where the $noteURL stands for a URL such as
sync:///Note/SyncNotes/BugReport/42:

   puts "Note Title: [note getprop $noteURL Title]"
```

note links

note links Command

NAME

```
note links          - Returns the set of links for a note
```

DESCRIPTION

```
This command returns information about notelinks. A notelink is a
relationship between a particular note and another object. The note
links command queries the database for notelinks that match certain
constraints or, if no constraints are given, all notelinks. The
constraints can be that the notelink be from a particular note, or
to a particular object, or both. Additionally, wildcarding is
supported to allow broadening the query to match groups of notes
or objects rather than specific notes and objects.

The 'note links' command with no arguments returns a list of
lists, where each sublist has two elements: the first is the object
URL, the second is the note URL.

With the -object option, only the URLs of notes linked to a given
object are returned.

With the -note option, URLs for all objects that are linked to
the specified note are returned.

This command is available only from server-side scripts.
```

SYNOPSIS

```
note links [-norec] [-note <noteURL> | -object <objURL>] [-pairs]
```

OPTIONS

-norec

```
-norec              This option must be used in conjunction with -object
                    and with wildcarding in <objURL>. It modifies the
                    meaning of the wildcard to prevent it from matching the
                    forward slash character (/), thus limiting the wildcard
                    to matching object URLs at the same hierarchical level as
                    the wildcard and not below.
```

-note

```
-note <noteURL>     Returns all of the objects that have
                    the specified note attached.

                    The <noteURL> argument is the note URL pattern
                    to match against in the query. In this URL, an
                    asterisk (*) may be used as a wildcard in place
                    of a note ID, to match any note of a particular
                    type, or an asterisk may be used instead of the
                    note type, the note ID, and the slash that would
                    separate them, to match any note of any type.
                    Valid URLs are of the form:

                    sync:///Note/SyncNotes/<notetypeName>/<id>
                    sync:///Note/SyncNotes/<notetypeName>/*
                    sync:///Note/SyncNotes/*

                    No other form of wildcarding is allowed; in
                    particular, a wildcard may not be used to match
                    part of a note ID, or part of a note type name,
                    or the word SyncNotes, or anything to the left
                    of SyncNotes.
```

-object

```
-object <objURL>    Returns all of the notes that are attached to
```

the specified object.

The <objURL> argument is the object URL pattern to match against in the query. In this URL, a trailing asterisk may be used as a wildcard to match attached objects whose URL begins with the specified pattern.

If the object URL pattern to be matched is a note, then the same wildcarding restrictions apply as for <noteURL>. Additionally, if the object URL to be matched is a user URL, only the trailing component (the user ID) may be wildcarded, and only in its entirety.

If the object URL pattern to be matched is not a note or user URL, slightly more flexible rules apply. The wildcard may still appear only as the end (last character), but unlike for <noteURL> it is allowed to match partial path elements. The pattern matching is strictly a substring search; the wildcard will match trailing @configname and ;versioned components and any number of embedded forward slashes (except when -norec is used).

This option must be used in conjunction with -object and with wildcarding in <ObjUrlPat>. It modifies the meaning of the wildcard to prevent it from matching the forward slash character (/), thus limiting the wildcard to matching object URLs at the same hierarchical level as the wildcard and not below.

This option forces the note links command to return a list of lists, even if -note, -object, or both, are specified. See the RETURN VALUE section for more detail.

-pairs

-pairs	This option forces the note links command to return a list of lists, even if -note, -object, or both, are specified. See the Return Value section for more detail.

RETURN VALUE

This command returns results in a variety of forms, depending on the command options you specify. In general, the output is whatever was not specified as an input:

1. If neither -object nor -note are used, then return all notelinks as a list of lists, with each sublist having an

```
   object URL and a note URL, in that order.

2. If -object <objURL> is used, then return a list of just the
   corresponding note URLs.

3. If -note <noteURL> is used, then return a list of just the
   corresponding object URLs.

4. If both -note and -object are used, then return no URLs,
   only the number of matching notelinks.

5. If -pairs is specified, the output is in the form described
   in rule 1, taking precedence over rules 2, 3, and 4.

The behavior ensures that if you passed in a note URL or an
object URL, you get back its counterpart directly and do not
have to extract it from a sublist. However, when either <objURL>
or <noteURL> or both include wildcards, it is usually desirable
to get back entire notelinks, with both the constituent URLs for
each link. The use of -pairs forces this complete form of return
value.

Note: URLs for notetype snapshots contain the note type name
appended by _OLD.
```

SEE ALSO

```
note attach, note detach, server-side, rstcl
```

EXAMPLES

- Example Showing All the Notes Attached to a Project
- Example Showing The Objects to which a Specific Note is Attached

Example Showing All the Notes Attached to a Project

```
This example displays the notes that are attached to the Asic
project before and after attaching a new note.

  foreach note [note links -object sync:///Projects/Asic] {
      puts "$note<BR>"
  }
  puts "<BR>"
  note attach sync:///Note/SyncNotes/BugReport/2 sync:///Projects/Asic
  foreach note [note links -object sync:///Projects/Asic] {
      puts "$note<BR>"
  }
```

Example Showing The Objects to which a Specific Note is Attached

```
This example displays the objects to which BugReport #2 is
attached (the Asic project, and the Beta configuration of Asic):

  set objects [note links -note sync:///Note/SyncNotes/BugReport/1]
  foreach note $objects {
      puts "$obj<BR>"
  }

Running this script outputs the following:

  sync:///Projects/Asic
  sync:///Projects/Asic@Beta
```

note query

note query Command

NAME

```
note query          - Queries the note system and returns note URLs
                      and values
```

DESCRIPTION

```
This command allows general queries against the notes database and
returns note URLs and values

This command is available only from server-side scripts.
```

SYNOPSIS

```
note query [[-attached <ObjUrl> [-norec]]
           [[-dbquery  <dbase_expr>] | [-sqlquery <sql_expr>]]
           [-filter ViewNote | EditNote] [-select <PropertyList>]
           [-type <notetype>]
```

OPTIONS

- -attached
- -dbquery
- -filter
- -norec
- -select
- -sqlquery
- -type

-attached

-attached <ObjUrl>	Limits results to notes that are attached to objects whose URLs match <ObjUrl>. <ObjUrl> may contain a trailing wildcard (*) as specified for -object <ObjUrl> in the note links command. When used with RevisionControl notes, the -attached option applies to the Objects field.

-dbquery

-dbquery <dbase_expr>	Returns only notes that match this query string. Must be a valid dBase query. This query is converted to an equivalent SQL expression and used to query the database. Only a subset of dBase syntax is supported and dBase queries may be retired in a future release.

-filter

-filter	For access controls, accepts either the ViewNote or EditNote action and returns a list of the notes allowed for the specified action.

-norec

-norec	Affects the way <ObjUrl> wildcards work, as specified for the note links command. This option must be used in conjunction with the -attached option and with wildcards.

-select

-select <PropertyList>	Returns both the URLs of notes matching the query and, for each matching note, the value for each property named in <PropertyList>. When -select is specified, the return value is a list of lists. Within each sublist, the first element is

the URL of a note matching the criteria; the others are values in that note for each of the properties specified in <PropertyList>.

You can include the keyword @LINKS as a property in <PropertyList> to return the note's attachments as a list within the return value.

-sqlquery

-sqlquery <sql_expr>

Returns only notes that match this query string. Must be a valid SQL expression. The expression is used verbatim in the WHERE clause of an SQL SELECT statement. No error checking is performed on this expression; it is passed directly to the database.

To avoid conflicts with property names and SQL keywords, the column names in the database are formed by prefixing the string f_ to the note property name. Thus a query for notes whose ID number is less than 10 and whose Author is 'joe' must be written in a -sqlquery clause as:

f_Id<10 AND f_Author='joe'

This prefixing is not necessary for -dbquery syntax.

-type

-type <notetype>

Returns only notes of this type, which must exist. If not specified, then the query is applied to all note types.

RETURN VALUE

If -select is not specified, a list of URLs matching the query criteria.

If -select is specified, the return value is a list of lists.
If no notes match the query criteria, an empty list is returned.

SEE ALSO

```
url notes, server-side, rstcl
```

EXAMPLES

Example Showing a list of URLS for all Note Types

```
This example displays a list of URLs of all notes of all types:
  puts [note query]

The resulting HTML page for a server with two notetypes (BugReport
and Note) with two notes of each notetype is:
  {sync:///Note/SyncNotes/BugReport/1} {sync:///Note/SyncNotes/
   BugReport/2} sync:///Note/SyncNotes/Note/1
   sync:///Note/SyncNotes/Note/2
```

Example Displaying a Specific Note Type Attached to a Project

```
The following example displays only notes of type Note that are
attached to the Asic project:
  puts [note query -type Note -attached sync:///Projects/Asic]

The resulting HTML page displays the URL of the only note to match
the query:
  sync:///Note/SyncNotes/Note/1
```

Example Returning Notes Created by a Specific User

```
This example returns a list of every note of any type that was
entered by sal, and then displays an pHTML page with the NoteID and
Title of each note:

  set noteList [note query -dbquery Author='sal']
  foreach noteUrl $noteList {
     url properties $noteUrl noteProps
     puts "NoteID $noteProps(Id) => $noteProps(Title)<BR>"
  }
```

Example Returning Notes Attached to a Specific Project

```
This example returns a list of every note that is attached to
```

```
objects under MyProj (including MyProj itself):
  set urls [note query -attached sync:///Projects/MyProj*]
whereas this example excludes MyProj:
  set urls [note query -attached sync:///Projects/MyProj/*]

With the 2.5 release, ProjectSync introduced a client-server
database (PostgreSQL). Consequently, some pre-2.5 constructs
using the note query command or combinations of commands may suffer
from degraded performance. Note query constructs like the following
may perform more slowly than they did under the pre-2.5 database:

foreach note [note query ....] {
url getprop $note prop1
}

Such constructs should be changed to the following form:

note query -select {prop1} ....
```

note relink

note relink Command

NAME

```
note relink         - Moves note attachments from one object
                      to another
```

DESCRIPTION

```
This command moves all note attachments from one object to another,
including links saved for a snapshot. The command also can move
attachments to objects below the original source object to
corresponding objects below the destination object; the effect is
to maintain the same relative tree structure before and after the
operation by re-rooting the tree at the destination object.
If the destination object does not have a tree of objects below it
to match the origin object tree, the attachments are moved but
are broken.

This command typically is used when a ProjectSync/DesignSync project
is relocated (renamed). <FromObjURL> and <ToObjURL> cannot be note
or user URLs, which cannot be renamed. However, the objects can
refer to a note type when a note type is renamed.
```

SYNOPSIS

```
note relink <FromObjURL> <ToObjURL> [-norec]
```

OPTIONS

- -norec

-norec

```
-norec              Causes the command to operate on (re-link) only
                    notes attached directly to the <FromObjURL>
                    object, and not any notes attached to objects
                    below it.
```

OPERANDS

- From Object URL
- To Object URL

From Object URL

```
<FromObjURL>        The object to rename. Must be a valid URL, but
                    may not reference a note or user object.
```

To Object URL

```
<ToObjURL>          The object to change the references to. Must be
                    a valid URL, but may not be a note or user object.
```

RETURN VALUE

```
The number of links that were updated.
```

SEE ALSO

```
note attach, note detach, note links, notetype rename
```

EXAMPLES

```
Move all the attachments from the Maine project to the Indiana
project:

note relink sync:///Projects/Maine sync:///Projects/Indiana
```

note schema

note schema Command

NAME

```
note schema          - Extracts information about a note type's
                       structure
```

DESCRIPTION

```
This command provides programmatic (stcl) access to the schema
that defines a note type.

This command is a wrapper to the notetype schema command and is
available for backward compatibility only. We discourage the use
of this command, as it may be dropped in future releases.

This command is available only from server-side scripts.
```

note setprops

note setprops Command

NAME

```
note setprops        - Sets property values on a note
```

DESCRIPTION

```
This command sets one or more property values (database fields)
on a note. This command gives you programmatic (stcl) access to the
note-editing capabilities of the ProjectSync graphical interface.

The first argument must be the URL for an existing note. Remaining
arguments can be either:
   o One or more pairs of database field name / new value, or
   o A single list containing pairs of field name / new value pairs
The latter form is useful when you do not know ahead of time
what properties you will be setting; build a list of the same
form that would be used for the Tcl "array set" command and then
pass the list to note setprops. This list can be empty.

The property values you supply must be legal for the corresponding
property type. The new property values specified in this command
are checked against the current values of the property. If they are
the same, the new value is discarded. Therefore, if all the values
```

specified in the command match their current values, the entire command is ignored. The command always attempts to set the complete set of property values on the object. If one or more attempts fail because of invalid values, no change is committed and the entire set of invalid property values is returned in the resultant error.

If a property name is specified multiple times with different values, the last value is stored.

It is not possible to change a note's ID number after the note is created.

The successful execution of this command causes an atomic note modify event and fires the corresponding triggers in response. If no property value is changed by the execution of the command, no event is generated.

When setting multiple properties on the same note, it is better to set them all in a single call to "note setprops" so that trigger activity is reduced.

The related "url setprop" command is limited to one name/value pair, but can operate on any object type, not just notes.

This command is available only from server-side scripts.

SYNOPSIS

```
note setprops [--] <note_url> <propname> <propvalue>
              [<propname> <propvalue>]...
note setprops [--] <note_url> <proplist>
```

OPTIONS

- --

--

```
--            Indicates that the command should stop looking for
              command options. Use this option when property
              names or values begin with a hyphen (-).
```

OPERANDS

- Note URL
- Property Name
- Property Value
- Property List Name/Value Pairs

Note URL

```
<note_url>    The URL of a note, which must exist, for which to
              set property values.
```

Property Name

```
<propname>    The name of a property on the note, which must exist
              on the note's note type.
```

Property Value

```
<propvalue>   A value to set the property of the note to, which
              must be a valid value for the property type.
```

Property List Name/Value Pairs

```
<proplist>    A list of property name and value pairs, in the style
              of Tcl's array set command - for example: {name1 val1
              name2 val2 ...}. The names must all be valid property
              names on the note type and the values must all be
              legal values for the property type.
```

RETURN VALUE

```
none
```

SEE ALSO

```
note getprop, url setprop, url getprop, url properties, server-side,
rstcl
```

EXAMPLES

- Example of Setting the Title on a Specific Note
- Example of Setting the Title and History for a Specific Note
- Example of Setting Various Properties on Specific Note

Example of Setting the Title on a Specific Note

```
This example sets the title on SyncDefect 42:
```

```
note setprops sync:///Note/SyncNotes/SyncDefect/42
    Title "A test"
```

Example of Setting the Title and History for a Specific Note

```
This example sets the title and entire history on SyncDefect 42:

note setprops sync:///Note/SyncNotes/SyncDefect/42 \
      Title "A test" Body "This is a test"
```

Example of Setting Various Properties on Specific Note

```
This example sets several properties on SyncDefect 42, using a
property list:

set newvalues(Resp) sal
set newvalues(State) closed
set newvalues(Priority) low

note setprops sync:///Note/SyncNotes/SyncDefect/42 [array
       get newvalues]
```

note systems

note systems Command

NAME

```
note systems        - Gets a list of note systems
```

DESCRIPTION

```
A list of all note systems on the server. If no note systems exist,
an empty list is returned.

Currently, only a single note system is defined, SyncNotes. This
note system always exists.
```

SYNOPSIS

```
note systems
```

RETURN VALUE

```
A list of all note systems on the server. Currently, this command
always returns a single item, SyncNotes.
```

EXAMPLES

```
Returns the list of note systems on the server:

note systems
```

note types

note types Command

NAME

```
note types          - Gets a list of defined note types for all
                      note systems
```

DESCRIPTION

```
Returns a list of note type URLs for all note types of all note
systems on the server. If no note types exist, an empty list is
returned.

This command is a wrapper to the notetype enumerate command and is
available for backward compatibility only. Synchronicity discourages
the use of this command, which may be dropped in future releases.
```

Note Type Manipulation

note types

note types Command

NAME

```
note types          - Gets a list of defined note types for all
                      note systems
```

DESCRIPTION

```
Returns a list of note type URLs for all note types of all note
systems on the server. If no note types exist, an empty list is
returned.

This command is a wrapper to the notetype enumerate command and is
available for backward compatibility only. Synchronicity discourages
the use of this command, which may be dropped in future releases.
```

notetype

notetype Commands

NAME

```
notetype            - Server-side commands to manipulate note types
```

DESCRIPTION

```
The 'notetype' family of commands provides access to the notetype web
object type. URLs for notetypes have the following form:

   sync:///Note/SyncNotes/<notetype>

 For example, the following URL specifies the BugReport notetype:

   sync:///Note/SyncNotes/BugReport

Notes can only be accessed from server-side scripts, so always use
the sync:/// syntax (no <host>:<port> specification).
```

SYNOPSIS

```
notetype <notetype_command> [<notetype_command_options>]

Usage: notetype [create|delete|enumerate|getdescription|rename|
                 schema]
```

EXAMPLES

```
See specific "notetype" commands.
```

notetype create

notetype create Command

NAME

```
notetype create     - Creates a new note type
```

DESCRIPTION

```
This command creates a new note type with the name you
specify. The properties Id, Title, Body, DateCreate, and Author
are automatically defined for all new note types.

You also can specify additional properties as a list of lists.
Each sublist within the property list defines an individual
property. A property definition consists of the following five
pieces of information, all of which must be specified: property
name, prompt string, IsRequired, property type, default value.

If you define other properties for the note type, you must specify
order-dependent values for each property. If you do not specify
additional properties, you must supply an empty Tcl list {}
following the NoteTypeName option.

This command is server-side only.
```

SYNOPSIS

```
notetype create [-description <DescStr>] [--]
<NotetypeName> {
               [{<PropertyName> <PromptName> <IsRequired>
               <PropertyTypeName> <DefaultValue>}...]
                }
```

OPTIONS

- -description
- --

-description

```
-description      Specifies a description for the note type; if
   <DescStr>      not specified, the description defaults to the
                  name of the note type. Enter the description in
                  quotation marks or curly braces following the
                  -description option. The description is limited
                  to 256 characters.
```

--

--	Indicates that the command should stop looking for command options. Use this option when an argument begins with a hyphen (-).

OPERANDS

Note Type Name

<NoteTypeName>	A valid name that you assign to the note type. A note type name is limited to 24 characters. Spaces are not allowed in note type names, and the legal character set for note type names consists of alphanumerics, hyphens, and underscores. The first character in a note type name cannot be a hyphen or a number.

Property Name

<PropertyName>	A unique name that you assign to the property. A property name is limited to 24 characters. Spaces are not allowed in property names, and the legal character set for property names consists of alphanumerics and underscores.

Prompt Name

<PromptName>	The prompt displayed on the GUI for users. Use only alphanumeric characters, not special characters or punctuation marks.

Is Required

<IsRequired>	A Tcl Boolean indicating whether a value for the property is required or optional. This setting is enforced only at the application level.

Property Type Name

```
<PropertyTypeName>  The name of a predefined property type. You can
                    use either one of ProjectSync's predefined
                    property types (such as Boolean, String80, Date)
                    or a property type you have defined yourself
                    using the Note Type Manager on the ProjectSync
                    GUI.
```

Default Value

```
<DefaultValue>      The default value for this property. If there is
                    no default value, specify an empty string "" as
                    a placeholder; the default value will be supplied
                    by the system. If you specify a default value, it
                    must be one of the valid values for this
                    property type. For example, if you have
                    defined a Choice property type, the default
                    value must be in your choice list.

                    The default value for a property in a note type
                    may be given as an empty string, meaning no
                    default. In this case, and when a note is created
                    with the note create command without a value for
                    the corresponding property, the property in the
                    note will remain unset.
```

RETURN VALUE

```
none
```

SEE ALSO

```
notetype delete, notetype getdescription, notetype rename,
server-side, rstcl
```

EXAMPLES

```
The following example creates a note type named BugReport, used
for defect tracking. In addition to the standard built-in property
set, this note type defines four additional properties: a required
user list field, Resp; a required State field; a required Severity
field; and an optional CC list.

notetype create BugReport -description "Defect tracking" \
  {{Resp Responsible 1 SyncUserList ""} \
```

```
{State State 1 BR_State new} \
{Severity Severity 1 BR_Severity serious} \
{cclist CCList 0 String240 ""}}
```

notetype delete

notetype delete Command

NAME

```
notetype delete     - Deletes the specified note type
```

DESCRIPTION

```
This command deletes the specified note type. You can specify only
one note type at a time. Any internal links to and from the
note type and any snapshots of the note type also are removed.
However, this command does not remove any triggers or customization
files associated with the note type.

Important: It is strongly recommended that you do not delete the
RevisionControl note type, which is a standard part of
ProjectSync. This note type is designed to work with DesignSync
for projects under revision control. Removing this note type
could cause problems if you later want to use ProjectSync notes
with DesignSync.

If the note type contains any notes, the -purgenotes option must
be specified to confirm your intent to delete the note type
(analogous to requiring rm -r for a nonempty directory).

This command is server-side only.
```

SYNOPSIS

```
notetype delete [-purgenotes] <NoteTypeName>
```

OPTIONS

- -purgenotes

-purgenotes

```
-purgenotes        Forces the deletion of all notes attached
                   to the specified note type. If you do not
                   specify this option and the note type you
```

want to delete contains notes, you get an error and the note type is not deleted.

OPERANDS

- Note Type Name

Note Type Name

<NoteTypeName>	The name of the note type to delete, which must exist.

RETURN VALUE

none

SEE ALSO

notetype create, notetype rename

EXAMPLES

The following example deletes the SyncDefect note type, including all notelinks, even if some notes exist for the note type:

```
notetype delete SyncDefect -purgenotes
```

notetype enumerate

notetype enumerate Command

NAME

notetype enumerate - Gets a list of defined note types for all note systems

DESCRIPTION

Returns a list of note type names for all visible note types of all note systems on the server. If no note types exist, an empty list is returned.

SYNOPSIS

```
notetype enumerate [-dbtablenames <dbTablenameVar>] [-urls]
```

OPTIONS

- -dbtablenames
- -urls

-dbtablenames

```
-dbtablenames          Store a map of the SQL table names by note
  <dbTablenameVar>     type name in the Tcl array named
                       <dbTablenameVar>. The table name is the
                       note type name prefixed by t_, but this
                       convention may not be used in the future. If
                       a note type name contains a hyphen, the
                       hyphen is converted to an underscore in the
                       table name. The information from this option
                       can be used to construct SQL statements
                       dynamically.
```

-urls

```
-urls                  The return list is formatted as note type
                       URLs instead of the note type names. This
                       output format is compatible with the format
                       from the note types command.
```

RETURN VALUE

```
A list of all note type names for all note systems.
```

SEE ALSO

```
note systems, notetype create, notetype delete, notetype rename,
url contents
```

EXAMPLES

```
Returns the list of note types on the server:

puts [notetype enumerate]
SyncDefect SW-Defect-1
```

notetype getdescription

notetype getdescription Command

NAME

```
notetype getdescription - Returns a brief description of the
                          note type
```

DESCRIPTION

```
Returns a brief description of the note type. The description
was set when the note type was created.
```

SYNOPSIS

```
notetype getdescription <NotetypeName>
```

OPERANDS

- Note Type Name

Note Type Name

```
<NotetypeName>            The name of the note type, which must
                          exist. <NotetypeName> is case-sensitive.
```

RETURN VALUE

```
A string containing the value of the brief description recorded
when the note type was created.
```

SEE ALSO

```
notetype create, url setprop
```

EXAMPLES

```
Returns the description of the SyncDefect note type:

notetype getdescription SyncDefect
```

notetype rename

notetype rename Command

NAME

```
notetype rename     - Renames an existing note type
```

DESCRIPTION

```
Renames an existing note type from <CurrentName> to <NewName>. The
note type specified by <NewName> must not already exist.

Any notelinks associated with the note type being renamed are also
converted to be associated with the renamed note type name. This
includes attachments both to and from the note type. Any snapshot of
the note type also is renamed.

The note-type-specific files in
$SYNC_CUSTOM_DIR/servers/<host>/<port>/share/data are not renamed as
part of the operation. However, these files are renamed if you
use the ProjectSync Note Type Manager to rename the note type.

This command is server-side only.

Important: It is strongly recommended that you do not rename the
RevisionControl note type, which is a standard part of
ProjectSync. This note type is designed to work with DesignSync
for projects under revision control. Renaming this note type
could cause problems if you later want to use ProjectSync notes
with DesignSync.
```

SYNOPSIS

```
notetype rename <CurrentName> <NewName>
```

OPTIONS

```
none
```

OPERANDS

- Current Name
- New Name

Current Name

<CurrentName>	The existing note type name that you want to change.

New Name

<NewName>	The new, legal note type name that you want to use. A note type name is limited to 24 characters. Spaces are not allowed in note type names, and the legal character set for note type names consists of alphanumerics, hyphens, and underscores. The first character in a note type name cannot be a hyphen or a number.

RETURN VALUE

none

SEE ALSO

note relink, notetype create, notetype delete

EXAMPLES

The following example changes the name of the note type from AcmeBug to AjaxBug:

```
notetype rename AcmeBug AjaxBug
```

notetype schema

notetype schema Command

NAME

```
notetype schema     - Extracts information about a note type's
                      structure
```

DESCRIPTION

```
This command provides programmatic (stcl) access to the schema that
defines a note type.

The base set of information provided by this command is the list of
fields that make up the note type. This information is provided in the
return value of the command. The various command-line options allow for
retrieving additional attribute information about each property on the
note type. The results for each type of property information are
returned in Tcl arrays that are passed in by name. The Tcl arrays need
not exist prior to the execution of the command. If the arrays do exist
or are of a scalar variable type, they are first cleared of all
information. The data returned in these Tcl arrays is indexed by
property name.
```

SYNOPSIS

```
notetype schema <NotetypeName> [-dbcolumns <ColumnsVar>]
                [-defaults <DefaultsVar>] [-notesys <NoteSystemName>]
                [-prompts <PromptsVar>] [-ptypes <TypesVar>]
                [-required <ReqdVar>]
```

OPTIONS

- -dbcolumns
- -defaults
- -notesys
- -prompts
- -ptypes
- -required

-dbcolumns

```
-dbcolumns          Stores a map of column names by field name in
  <ColumnsVar>      the Tcl array named <ColumnsVar>. Currently, the
                    column name is always the field name prefixed by
                    f_, but this convention may not be used in the
                    future. This information from this option can be
                    used to construct SQL statements dynamically.
```

-defaults

```
 -defaults            Stores a map of default values by field name in
  <DefaultsVar>       the Tcl array named by <DefaultsVar>
```

-notesys

```
 -notesys             A valid note system name. Defaults to SyncNotes.
  <NoteSystemName>    (This name is an input parameter, not an output
                      array name.)
```

-prompts

```
 -prompts             Stores a map of field prompt strings by field
  <PromptsVar>        name in the Tcl array named by <PromptsVar>
```

-ptypes

```
 -ptypes              Stores a map of property type names by field
  <PtypesVar>         name in the Tcl array named by <PtypesVar>
```

-required

```
 -required            Stores a map of required flags (1 or 0) by field
  <ReqdVar>           name in the Tcl array named by <ReqdVar>
```

RETURN VALUE

```
A list of all field names in the specified note type.
```

SEE ALSO

```
note getprop, notetype create, url getprop, url properties
```

EXAMPLES

- Example Returning all Fields in the Specified Note Type
- Example Displaying the Types for Each Field

Example Returning all Fields in the Specified Note Type

```
This example returns all the fields in the SyncDefect note type:

set field_names [notetype schema SyncDefect]
```

Example Displaying the Types for Each Field

```
This example displays the type of each field:

set field_names [notetype schema -ptypes types SyncDefect]
foreach field $field_names {
   puts "$field: $types($field)<BR>"
}

The above example generates output such as this:

KeyWords: String80
Browser: SD-Browser
Title: String80
Class: SD-Class
Platform: SD-Platform
Customer: SD-Customers
DateCreate: Timestamp

This example determines whether the FixDate field in a note type
called ECO is required:

notetype schema -required reqd ECO
if {$reqd(FixDate)} { puts "FixDate is required." }
```

Property Type Information

ptype

ptype Commands

NAME

```
ptype               - Commands that get information about property types
```

DESCRIPTION

```
The ptype commands return information about property types.
Property types are the data types available for note type
properties (fields). When you create a note type field
such as  'SpecAuthor', you assign it a property type, such as
'String80'. Synchronicity provides a number of predefined
property types, or you can create your own. You create and
modify property types from the Property Type Manager
```

```
(accessed from the Note Type Manager in ProjectSync's graphical
user interface). See the ProjectSync User's Guide for
more information on property types.

The ptype commands are not server-side only, but are typically
used when accessing note and note-type objects, which are
only accessible from server-side scripts.
```

SYNOPSIS

```
ptype <ptype_command> [<ptype_command_options>]

Usage: ptype [choices|class|enumerate|is|strwidth|transitions]
```

OPTIONS

```
Vary by command.
```

RETURN VALUE

```
Varies by command.
```

SEE ALSO

```
ptype choices, ptype class, ptype enumerate, ptype is,
ptype strwidth, ptype transitions
```

EXAMPLES

```
See specific "ptype" commands.
```

ptype choices

ptype choices Command

NAME

```
ptype choices       - Returns the set of legal values for
                      an enumerated type
```

DESCRIPTION

```
This command returns the set of legal values for an existing choice
or state machine property type. For a state machine, the set of
choices returned is the entire list of possible state values. Only
choice and state machine property types may be specified. It is an
error to supply a property of any other type.
```

SYNOPSIS

```
ptype choices <ptype_name>
```

OPERANDS

- Custom Property Type Name

Custom Property Type Name

```
<ptype_name>      The name of an existing choice or state machine
                  custom property type.
```

RETURN VALUE

```
The list of legal values for the property type.
```

SEE ALSO

```
ptype is, ptype class, ptype transitions
```

EXAMPLES

```
This examples shows 'ptype choices' applied to several property
types:
  puts [ptype choices SyncPriority] # choice class
  => low medium high stopper
  puts [ptype choices SyncState]    # state-machine class
  => open analyzed fixed closed
```

ptype class

ptype class Command

NAME

```
ptype class          - Returns the class of a property type
```

DESCRIPTION

```
This command returns the class of a property type, where a class
is a general category of property types. For example, String10,
String80, and String are all property types of the 'string'
class. Note that state-machine property types belong to both the
'machine' and 'choice' classes; the dominant class is 'machine'
and is therefore returned by 'ptype class'.
```

SYNOPSIS

```
ptype class <ptype_name>
```

OPERANDS

- Custom Property Type Name

Custom Property Type Name

```
<ptype_name>         The name of an existing property type.
```

RETURN VALUE

```
Returns one of the following strings:
  boolean   - for boolean types
  choice    - for choice (enumeration) types
  date      - for Date types
  machine   - for state machine types
  number    - for integer, float and other numeric types
  string    - for string types (of any size)
  time      - for Time types
  timestamp - for Timestamp types
  userlist  - for user list types
```

SEE ALSO

```
ptype choices, ptype enumerate, ptype is
```

EXAMPLES

```
This example shows 'ptype class' applied to several property
types:
  puts [ptype class String80]
  => string
  puts [ptype class String]
  => string
  puts [ptype class SyncState]
  => machine
```

ptype enumerate

ptype enumerate Command

NAME

```
ptype enumerate     - Returns a list of all property types
```

DESCRIPTION

```
Generates a list of all property types on the server. The set of
property types returned consists of all custom choice and state
machine property types plus the built-in base property types:

String          Boolean       SyncClass
String10        Integer       SyncUserList
String20        Float         SyncPriority
String80        Date          SyncRevCtrlCmd
String240       Time          SyncState
String512       Timestamp
String4000b     "WebObject"

This list excludes the string-derived property types like cclist,
keywords, and fileattach. These are all defined, and their semantics
imposed, at the note panel level.
```

SYNOPSIS

```
ptype enumerate
```

OPTIONS

```
none
```

RETURN VALUE

```
List of property types.
```

EXAMPLES

```
This example lists all the property types known to the server:

  puts [ptype enumerate]
  => String80 String Boolean String240 Integer SyncClass String20
     Float String10 SyncRevCtrlCmd {WebObject } String512 SyncPriority
     Time Timestamp Date String4000b SyncUserList SyncState
```

ptype is

ptype is Command

NAME

```
ptype is            - Tests whether a property type is of a
                      certain class
```

DESCRIPTION

```
This command tests whether the specified property type is of the
specified class, where a class is a general category of property
types. For example, String10, String80, and String are all
property types of the 'string' class. Note that state-machine
property types belong to both the 'machine' and 'choice' classes.
```

SYNOPSIS

```
ptype is <[-boolean] | [-choice] | [-date]
       | [-machine] | [-number] | [-string]
       | [-time] | [-timestamp] [-userlist]>
        <PropertyTypeName>
```

OPTIONS

- -boolean
- -choice
- -date
- -machine
- -number
- -string

-boolean

```
 -boolean    Check if the property type is a boolean.
```

-choice

```
 -choice     Check if the property type is a choice (enumeration)
             type. Note that 'ptype is -choice' also returns 1
             (TRUE) if the property type is a state machine.
```

-date

```
 -date       Check if the property type is a date.
```

-machine

```
 -machine    Check if the property type is a state machine.
```

-number

```
 -number     Check if the property type is an integer, float, or
             other numeric type.
```

-string

```
 -string     Check if the property type is a string (of any size).
```

-time

```
 -time       Check if the property type is a time.
```

-timestamp

```
 -timestamp  Check if the property type is a timestamp.
```

-userlist

```
-userlist   Check if the property type is a user list.
```

OPERANDS

- Custom Property Type Name

Custom Property Type Name

```
<PropertyTypeName>  The name of an existing property type
```

RETURN VALUE

```
Returns 1 (TRUE) if the property type is of the specified class;
0 (FALSE) otherwise.
```

SEE ALSO

```
ptype class, ptype enumerate
```

EXAMPLES

```
This example verifies that the SyncPriority property type is of
the 'choice' class (and not of the 'string' class).

  puts [ptype is -choice SyncPriority]
  => 1
  puts [ptype is -string SyncPriority]
  => 0
```

ptype strwidth

ptype strwidth Command

NAME

```
ptype strwidth      - Returns the maximum width for strings of this type
```

DESCRIPTION

```
This command returns the maximum number of characters a string-based
property type can hold. The property type specified must be based on
a string class. This command deals with character width, not byte width.
```

SYNOPSIS

```
ptype strwidth <ptype_name>
```

OPERANDS

- Custom Property Type Name

Custom Property Type Name

```
<ptype_name>        The name of an existing property type.
```

RETURN VALUE

```
Returns the maximum number of characters allowed for the given property
type. For the String property type, -1 is returned, indicating no
maximum.
```

SEE ALSO

```
ptype is, ptype class
```

EXAMPLES

```
This example shows 'ptype strwidth' applied to several
property types.
  puts [ptype strwidth String80]
  => 80
  puts [ptype strwidth String10]
  => 10
  puts [ptype strwidth String]
  => -1
  puts [ptype strwidth Boolean]
  => Boolean: not a string type
```

ptype transitions

ptype transitions Command

NAME

```
ptype transitions   - For the value of a state machine, returns
                      the set of values that are legal for that
                      value to change to
```

DESCRIPTION

```
This command returns a list of the valid next states for any
specified state of a State Machine property type. The -from
option is required and indicates the state for which you want to
know all possible next states.
```

SYNOPSIS

```
ptype transitions <PropertyTypeName> <-from <PropertyValue>>
```

OPTIONS

- -from

-from

```
-from               Returns the list of states that are valid from
  <PropertyValue>   <PropertyValue>, which must be a legal value for
                    the state machine.
```

OPERANDS

- Custom Property Type Name

Custom Property Type Name

```
<PropertyTypeName> The name of an existing property type.
```

RETURN VALUE

```
A list of legal state values from the value specified.
```

SEE ALSO

```
ptype choices, ptype is
```

EXAMPLES

```
The following example returns a list of all the valid next states for the
state 'fixed' of the SyncState property type.

  puts [ptype choices SyncState]
  => open analyzed fixed closed
  puts [ptype transitions SyncState -from fixed]
  => open closed
```

Email Subscription Manipulation

subscription

subscription Commands

NAME

```
subscription        - Commands to manipulate email subscriptions
```

DESCRIPTION

```
These commands allow you to manage ProjectSync email subscriptions
which let you be notified when certain kinds of activity, such as
revision control operations, or defect tracking, take place in
DesignSync or ProjectSync.
```

SYNOPSIS

```
subscription <subscription_command> [<subscription_command_options>]

Usage: subscription [add|delete|edit|get|list]
```

OPTIONS

```
Vary by command.
```

subscription add

subscription add Command

NAME

```
subscription add    - Subscribes for email related to specified
                      objects
```

DESCRIPTION

```
This command has two basic forms. The first form of this command is
general. It allows you to subscribe for email notifications related
to any note type on the server managing the vaults for the specified
objects.

The second form of this command is specific to RevisionControl
notes.  It allows you to subscribe for email notifications related
to RevisionControl notes on the server managing the vaults for the
specified objects. This form also has convenient options for
limiting the subscription to certain RC operations.

Note that a single 'subscription add' command invocation can cause
multiple entries in the subscription database, as reported by
the 'subscription list' command. For example, this command
creates two entries:
    subscription add *.tcl *.html
and this one creates four entries:
    subscription add -ci -tag *.tcl *.html

Note:  If you specify a subscription for the tag command, but do not have a
subscription for the ci command, you will not get notifications for
any ci -tag operations that specify a tag.  In order to get all tag
activity, you must also subscribe to ci command notifications.
```

SYNOPSIS

```
subscription add [-ci] [-colock] [-conolock] [-filter Expr]
                 [-noteType noteType] [-server serverURL] [-tag]
                 [-tagname TagName] [-unlock] [-user user] object...
```

ARGUMENTS

- Object

Object

```
 object    An expression (possibly containing wildcards) that must
           match one of the objects associated with a note in order
```

```
            for you to receive email. If no objects are provided, the
            default is to subscribe for ALL objects.
```

OPTIONS

-ci

```
 -ci        Indicates that you wish to receive email when the objects
            listed are checked in. Implies the note type
            'RevisionControl'.
```

-colock

```
 -colock    Indicates that you wish to receive email when the objects
            listed are checked out with a lock.  Implies the note type
            'RevisionControl'.  The same assumptions are applied for
            'populate -lock' operations.
```

-conolock

```
 -conolock  Indicates that you wish to receive email when the
            objects listed are checked out without a lock.  For the
            'co -get' and 'populate -get' operations, the note type
            'RevisionControl' is assumed.
```

-filter

```
 -filter    Gives an expression describing which modifications to
            notes of the given type(s) will cause email to be
            sent. For a complete explanation of the filter
            expression syntax, see the ProjectSync
            User's Guide 'Advanced Email Subscriptions' topic. Note
            that if stcl is used the '$' operator needs to be
            escaped when used in a filter. For example, -filter
```

```
            Title\$Test.
```

-notetype

```
 -noteType  Specifies the name of the note type for which subscriptions
            are to be added.

            If -ci, -conolock, -colock, -unlock, or -tag are used, then
            the noteType is assumed to be 'RevisionControl'.

            If a note type is not provided and there are not any revision
            control operations listed, subscriptions for ALL note
            types will be modified.
```

-server

```
 -server     Gives the URL of the server to query for subscriptions.
            If no server URL is given, the server owning the vault for
            the current working directory is assumed.
```

-tag

```
 -tag       Indicates that you wish to receive email when the objects
            listed are tagged. Implies the note type
            'RevisionControl'.
```

-tagname

```
 -tagName   Limits your email subscriptions to those concerning
            revision control operations associated with the
            given tag.
```

-unlock

```
 -unlock    Indicates that you wish to receive email when the
            objects listed are unlocked.  Implies the note type
            'RevisionControl'. The same assumptions are applied for
            'cancel' operations.
```

-user

```
 -user      Allows you to add subscriptions for another user. By
            default, subscriptions are shown for the user running
```

```
            the client.
```

RETURN VALUE

```
none
```

SEE ALSO

```
subscription delete, subscription edit, subscription get,
subscription list
```

EXAMPLES

Example of Subscribing to SyncDefect Notes

```
This example shows subscribing to SyncDefect notes attached to any
object under project ASIC on the SyncServer.

dss> subscription add -noteType SyncDefect sync:///Projects/ASIC
```

Example of Subscribing to Specific Objects Tagged with a Specific Tag

```
This example shows two different ways to subscribe to RevisionControl
notes when objects representing the .tcl files under [url vault .]
are tagged as golden.

dss> subscription add -noteType RevisionControl -filter \
    "Command=tag;Tag=Golden" *.tcl

dss> subscription add -tag -tagname Golden *.tcl
```

Example of Subscribing to Specified RC Notes

```
Subscribe to RC notes for objects *.tcl and *.ini for operations:
checkin, checkout with lock.
```

```
dss> subscription add -ci -colock *.tcl *.ini
```

Example of Subscribing to all Notes Attached to a Module (Module-based)

```
This example subscribes to all notes attached to a module.

dss> subscription add -server sync://serv1.ABCo.com:2647 \
     sync:///Modules/ChipDesign/NXZ-21x
```

Example of Subscribing to all Notes for Modules in a Category (Module-based)

```
dss> subscription add -server sync://serv1.ABCo.com:2647 \
     sync:///Modules/ChipDesign
```

Example of Subscribing to all Notes Attached to a Project (File-based)

```
This example subscribes to all notes attached to objects
sync:///Projects/ASIC on the SyncServer.

dss> subscription add ASIC
```

subscription delete

subscription delete Command

NAME

```
subscription delete - Deletes email subscriptions for specified objects
```

DESCRIPTION

```
This command allows you to delete email subscriptions for the specified
objects. For each object you specify, the server hosting the vault for
that object will be searched for subscriptions.

Use the 'subscription list' command to view your existing
subscriptions.

This command provides several different ways to delete subscriptions.
 * Delete all subscriptions for a specified user on the server.
 * Delete any subscriptions which were defined to refer to ALL
   (objects).
 * Deletes any subscriptions for a specified note type which were
   defined to refer to ALL (objects).
```

```
  * Delete subscriptions for a user on a specified object, optionally
    for a specified note type.

 Examples of all of these methods are shown in the Examples section.
```

SYNOPSIS

```
 subscription delete [-noteType noteType] [-server serverURL]
                     [-user user] [<object>[ ...]]
```

ARGUMENTS

- Object

Object

```
  object     The objects for which subscriptions are to be deleted.
             For each object you specify, the server hosting the vault
             for that object is searched for subscriptions. You may use
             wildcards in the object value.

             Note: The object specification must match exactly the
             specification in the subscription database. For instance,
             if you subscribed for Projects/MyProj/Foo/bar, you cannot
             unsubscribe projects/MyProj/Foo/*  because that is not an
             exact match.

             If an object is not specified, subscriptions to ALL
             objects (those subscriptions for which the user selected
             ALL objects instead of specifying a particular object) are
             deleted.  If a noteType value is provided without an
             object, subscriptions to ALL objects for that noteType are
             deleted.
```

OPTIONS

- -notetype
- -server
- -user

-notetype

```
 -noteType  Specifies the name of the note type for which subscriptions
            are to be deleted.
```

-server

```
-server    Gives the URL of the server to query for subscriptions.
           If no server URL is given, the server owning the vault for
           the current working directory is assumed.
```

-user

```
-user      Allows you to delete the subscriptions of a user other
           than the user you are logged in as. By default, the
           name of the user running the client is used.
```

RETURN VALUE

```
none.  After the command is run, it returns a single integer value
showing how many subscriptions were deleted during the operation.
```

SEE ALSO

```
subscription add, subscription edit, subscription get, subscription list
```

EXAMPLES

- Example of Deleting all Subscriptions for a User
- Example of Deleting Subscriptions for all Objects
- Example of Deleting Subscriptions for a NoteType for all Objects
- Example of Deleting all Subscriptions on the Specified Object
- Example of Deleting all Specified Notetypes for an Object

Example of Deleting all Subscriptions for a User

```
This example shows deleting all subscriptions created for a specified
user on the specified server.
Note: By using the wildcard (*) for the object, you specify deleting
all subscriptions for all objects for the user on the server.
dss> subscription delete -user rsmith -server
     sync://srv1.ABCo.com:2647 *
 30
```

Example of Deleting Subscriptions for all Objects

```
This example shows deleting the subscriptions created to match all
objects (either by not specifying an object when the subscription was
added or by choosing the ALL option).
```

```
Note: No value, not even wildcard, is specified for object.

dss> subscription delete -user rsmith -server sync://srv1.ABCo.com:2647
 15
```

Example of Deleting Subscriptions for a NoteType for all Objects

```
This examples shows deleting the subscriptions for the Defect note type.

Note: By not specifying a user, the subscriptions deleted are those
of the user running the command.

dss> subscription delete -notetype Defect -server sync://srv1.ABCo.com:2647
 10
```

Example of Deleting all Subscriptions on the Specified Object

```
This example shows deleting all subscriptions for the specified user
for the specified project.
Note: The /// construction shows that it is an object,not a reference
to the server.

dss> subscription delete -user rsmith sync:///Projects/ProjectSync/*
30
```

Example of Deleting all Specified Notetypes for an Object

```
This example shows deleting all Defect note types for all objects
within the panels project.

dss> subscription delete -noteType Defect panels/*
8
```

subscription edit

subscription edit Command

NAME

```
subscription edit   - Edits email subscriptions
```

DESCRIPTION

```
This command invokes a web browser to edit the email subscription of
the given user.

On Unix platforms, the default browser chosen by your system
administrator will be used. To change the default, use the 'SyncAdmin'
tool. On Windows platforms, the default browser defined in the system
registry will be used.
```

SYNOPSIS

```
subscription edit [-server serverURL] [-user user]
```

OPTIONS

-server

```
-server    Gives the URL of the server to query for subscriptions.
           If no server URL is given, the server owning the vault for
           the current working directory is assumed.
```

-user

```
-user      Allows you to edit the subscriptions of a user other than
           the user you are logged in as. By default, the name of
           the user running the client is used.
```

RETURN VALUE

```
1
```

SEE ALSO

```
subscription add, subscription delete, subscription get,
subscription list
```

EXAMPLES

- Example of Editing a Subscription on a Specified Server

Example of Editing a Subscription on the Server Associated with cwd

```
This example shows the command that launches your web browser to edit
subscriptions on the server associated with the current working
directory, pointed to by [url vault .]

dss> subscription edit
```

Example of Editing a Subscription on a Specified Server

```
Brings up your web browser to edit subscriptions on SyncServer:2647.

dss> subscription edit -server sync://SyncServer:2647
```

subscription get

subscription get Command

NAME

```
subscription get    - Gets subscription information as a Tcl list
```

DESCRIPTION

```
For a user-friendly view of the subscriptions, use the 'subscription
list' command. This command is intended for use by those who wish to
manipulate the subscription database using Tcl.

Each subscription entry is returned as a Tcl list with alternating
names and values ('array get' format). The following names are used:

  noteType  The name of the NoteType for which the use is subscribed.
  object    The object for which the user is subscribed.
  filter    The filter string for the subscription.
```

SYNOPSIS

```
subscription get [-noteType noteType] [-server serverURL] [-user user]
```

OPTIONS

- -noteType

- -server
- -user

-noteType

```
-noteType  Specifies the name of the note type for which subscriptions
           are to be queried. If no note type is given,
           subscriptions for all note types are listed.
```

-server

```
-server    Gives the URL of the server to query for subscriptions.
           If no server URL is given, the server owning the vault for
           the current working directory is assumed.
```

-user

```
-user      Allows you to get the subscriptions of a user other than
           the user you are logged in as. By default, the name of
           the user running the client is used.
```

RETURN VALUE

```
A Tcl list of subscriptions, where each subscription is represented
in Tcl 'array get' format, with the array indices being 'noteType',
'object', and 'filters'.
```

SEE ALSO

```
subscription add, subscription delete, subscription edit,
subscription list
,
```

EXAMPLES

```
foreach sub [subscription get -user JDoe] {
   array set info $sub
   puts "NoteType: $info(noteType)"
   puts "Object:   $info(object)"
   puts "Filter:   $info(filters)"
 }

The output would resemble the following:
```

```
NoteType: RevisionControl
Object:   *.tcl
Filter:   Tag=Golden;Command=tag
```

subscription list

subscription list Command

NAME

```
subscription list   - Lists email subscriptions
```

DESCRIPTION

```
This command prints out a listing of your current subscriptions
on a particular server.

Note that a single 'subscription add' invocation can cause
multiple entries in the subscription database, as reported by
the 'subscription list' command. For example, this command
creates two entries:
    subscription add *.tcl *.html
and this one creates four entries:
    subscription add -ci -tag *.tcl *.htm
```

SYNOPSIS

```
subscription list [-noteType noteType] [-server serverURL] [-user user]
```

OPTIONS

- -noteType
- -server
- -user

-noteType

```
-noteType  Specifies the name of the note type for which subscriptions
           are to be displayed. If no note type is given,
           subscriptions for all note types are listed.
```

-server

```
-server    Gives the URL of the server to query for subscriptions.
           If no server URL is given, the server owning the vault for
           the current working directory is assumed.
```

-user

```
-user      Allows you to view the subscriptions of a user other than
           the user you are logged in as. By default, the name of
           the user running the client is used.
```

RETURN VALUE

```
none
```

SEE ALSO

```
subscription add, subscription delete, subscription edit,
subscription get
```

EXAMPLES

- Example Showing Listing Subscriptions on the server
- Example Showing Listing Subscriptions for Vault Associated with cwd

Example Showing Listing Subscriptions on the server

```
This example shows a list of all subscriptions for a specified
server.

dss> subscription list -server sync://SyncServer:2647

 SyncDefect  sync:///Projects/EmailPackage State->new;Class=enhancement
 SyncDefect  sync:///Projects/EmailPackage State->new;Class=sw-bug
 SyncDefect  sync:///Projects/ProjectSync  Class=sw-bug
 SyncDefect  sync:///Projects/ProjectSync  Severity->STOPPER
 HW-Defect-1 sync:///Projects/ProjectSync
 Note        sync:///Projects/ProjectSync
```

Example Showing Listing Subscriptions for Vault Associated with cwd

```
This example shows a list of all the subscriptions for the vault
associated with the current working directory.  You could also
specify current working directory as "[url vault .]".
```

```
dss> subscription list -noteType Note
```

User Profile Manipulation

user

user Commands

NAME

```
user                - Server-side commands to edit user profiles
```

DESCRIPTION

```
The 'user' family of commands lets you access SyncServer user
profiles. You can add and delete user IDs and profiles.
```

SYNOPSIS

```
user <user_command> [user_command_options>]

Usage: user [counts|create|delete]
```

EXAMPLES

```
See specific "user" commands.
```

user counts

user counts Command

NAME

```
user counts         - Counts the number of user records
```

DESCRIPTION

```
Counts the number of currently defined user records. This command
```

```
is equivalent to length [url users sync:///], but is faster.
```

SYNOPSIS

```
user counts
```

OPTIONS

```
none
```

RETURN VALUE

```
Returns the number of currently defined user records.
```

SEE ALSO

```
note counts, user create
```

EXAMPLES

```
This command outputs the number of users:

puts [user counts]

25
```

user create

user create Command

NAME

```
user create         - Creates a new user id with the specified profile
```

DESCRIPTION

```
This command creates a new user ID and profile with the name, email
address and password that you specify. The user ID cannot already
exist. This command is available only from server-side scripts.
```

A username should consist of alphanumeric characters. Do not include spaces, single quotation marks ('), double quotation marks ("), leading dashes (-), dollar signs ($), ampersands (&), or slashes (/) in user names.

The user profile attributes are specified as a list of name/value pairs. The user create command accepts the following attribute names:

```
Name               - The user's name.
EmailAddr          - The user's email address.
ClearKey           - The user's password, in cleartext. The value is
                     encrypted using MD5 encoding.
Key                - The user's password, assumed to be in MD5 format.
                     The value is used as is.
PhoneNumbr         - The user's phone number.
PageNumber         - The user's pager number.
```

Values for the name, email address, and password attributes must be supplied.

For the password attribute, either ClearKey or Key may be used, but not both. If ClearKey is used, the value is assumed to be cleartext and is first encrypted using MD5 encoding. Use the ClearKey argument when the password is not encoded.

If Key is used, the value is assumed to be encrypted and is used as is. Use the Key argument when the user's password is encrypted with Unix style crypt() or when transferring user accounts, including passwords, from the Unix NIS database into ProjectSync. You can set the Key with url setprop and retrieve it with url getprop. ClearKey is write-only, so you can set it with url setprop but you cannot retrieve it with url getprop. For more information, see the Tcl Script for Importing Users and Changing User Passwords topics in the ENOVIA Synchronicity DesignSync Data Manager Administrator's Guide.

If the same property is specified more than once with different values, the last value is stored.

Note: If you use LDAP to store user information, you can use ProjectSync's LDAP client to give users from an LDAP database access to ProjectSync. For other user databases, you can create a trigger that fires whenever a user tries to access an area of ProjectSync that requires user authentication. See the topics Enabling LDAP and Creating User Authentication Scripts in the ENOVIA Synchronicity DesignSync Data Manager Administrator's Guide for more information.

SYNOPSIS

```
user create <userid> <{Name Value Name Value...}>
```

OPTIONS

```
none
```

OPERANDS

- User ID
- Name/Value Pairs for Attribute Values

User ID

```
<userid>            A unique user ID for new user profile.
```

Name/Value Pairs for Attribute Values

```
{Name Value         User profile attribute values, in array list
  Name Value...}    format. The names must match the existing
                    property names on the user profile and the
                    values must be legal for the property types.
```

RETURN VALUE

```
none
```

SEE ALSO

```
user delete, url properties, url getprop, url setprop
```

EXAMPLES

```
This example creates a user profile for the username jane. The
user's full name is Jane Doe, her email address is jane@hb.com
and her password is abcde.

user create jane {Name "Jane Doe" EmailAddr "jane@hb.com" ClearKey "abcde"}
```

user delete

user delete Command

NAME

```
 user delete          - Deletes the specified user ID and corresponding
profile
```

DESCRIPTION

```
 This command deletes a specified user ID and corresponding profile
 from the SyncServer. Additional user data associated with this profile,
 such as email subscriptions, are not modified by this command. The
 user interface performs these additional clean-up operations.

 This command is available only from server-side scripts.
```

SYNOPSIS

```
 user delete <username>
```

OPTIONS

```
 none
```

OPERANDS

- User ID

User ID

```
 <username>           The user ID of the user profile to be deleted,
                      which must exist.
```

RETURN VALUE

```
 none
```

SEE ALSO

```
 user create
```

EXAMPLES

In this example, you delete the username "jane" and corresponding profile from the SyncServer.

```
user delete jane
```

Getting Assistance

Using Help

ENOVIA Synchronicity DesignSync Data Manager Product Documentation provides information you need to use the product 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 DesignSync tools installation. See SyncAdmin Help for details.

This help is available as both as text only within the command line interface and the HTML guide, which is currently being viewed. The command line interface help displays in the standard output window.

To bring up the online help from the tool you are using, do one the following:

- Type **help <*command*>** from the tool you are using. This displays the full command description, usage (syntax), explanation of options, examples, and error messages. If the command contains a prefix, enclose the command in double quotes, for example: `help "url properties"` displays help for the `url properties` command.
- Type **<*command*> -help** from the tool you are using. This displays the full command description, usage (syntax), explanation of options, examples, and error messages.
- Type **<*command*> -usage** from the tool you are using. This displays the short command description and the command syntax.

Note: DesignSync also features a webhelp command in the command line interface, which opens the HTML page for the specified command.

When the HTML 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 givens you access to the Contents, Index, and Search navigation tools.
- **<<** and **>>**: Clicking these buttons moves you to the previous or next topic in a series within 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 *ENOVIA Synchronicity Command Reference* is available in book format from the ENOVIA Documentation CD or the DSDocumentationPortal_Server installation available on the 3ds support website (http://media.3ds.com/support/progdir/). 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.

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